VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Firmware pro systém správy a monitorování v automatizaci

Firmware for Remote Control and Monitoring Industrial Applications

2012 **Bc. Zbyněk Složil**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Zbyněk Složil**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Firmware pro systém správy a monitorování v automatizaci
Firmware for Remote Control and Monitoring Industrial Applications

Zásady pro vypracování:

Cílem diplomové práce je vyvinout firmware pro komunikační modul na bázi AT91SAM7X. Výsledná implementace poběží na real-time operačním systému FreeRTOS a umožní vzdálené monitorování naměřených veličin, popřípadě zprostředkuje řízení připojených zařízení. Systém komunikuje s uživatelem přes webové rozhraní nebo GSM bránu a zároveň zajišťuje sběr a ukládání monitorovaných dat. Součástí aplikace je i jednoduchá
grafická vizualizace monitorovaných hodnot.

1. Popište operační systém FreeRTOS a jeho API.
2. Obsluha I/O portů a jeho API.
3. Komunikace se síťovou ochranou a řídicími systémy.
4. Komunikace s GSM/GPRS modemem.
5. Zálohování dat na paměťové medium.
6. HTTP server s podporou POST/GET.
7. Grafická vizualizace naměřených dat.
8. Výsledky z testování v reálném provozu.

Seznam doporučené odborné literatury:

Richard Barry, Using the FreeRTOS Real Time Kernel - Standard Edition, 2010, ISBN 978-1446169148
Rob Williams, Real-Time Systems Development, Butterworth-Heinemann, 2005, ISBN 978-0750664714

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Michal Krumnikl**

Datum zadání: 18.11.2011

Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 4. května 2012 Zbyněk Složil

Rád bych tímto poděkoval Ing. Michalu Krumniklovi za odborné vedení mé diplomové práce a trpělivost, kterou mi laskavě věnoval.

Abstrakt

Práce se věnuje vývojem firmwaru pro monitorovací systém. Firmware běží na real-time operačním systému FreeRTOS. Jeho hlavní úlohou bude monitorování dat, které budou uživateli zobrazeny přes webové rozhraní. Data budou sbírány zařízením Síťová ochrana. Na případné chyby a důležité informace bude uživatel upozorněn pomocí SMS zprávy.

Klíčová slova:

ARM, AT91SAM7X, FreeRTOS, lwIP, Prevel, JSON, AJAX, HTTP server, SNTP klient, síťová ochrana, GSM modem

Abstract

This work deals with development firmware for monitoring system. Firmware works on real-time operating system FreeRTOS.. It`s main task is monitoring datas, which will be shown to the user through the web interface. Datas will be collecting with Network system protection. For eventually errors user will be warned by SMS message.

Keywords:

ARM, AT91SAM7X, FreeRTOS, lwIP, Prevel, JSON, AJAX, HTTP server, SNTP client, network system protection, GSM modem

Seznam použitých symbolů a zkratek

<CR> - Carriage Return, v ASCII hodnota 0x0d
<LF> - Line Feed, v ASCII hodnota 0x0a
AJAX - Asynchronous JavaScript and XML, technologie vývoje interaktivních webových aplikací
ARP - Address Resolution Protocol, používá se k získání Ethernetové MAC adresy v počitačových sítí
CISC - Complex Instruction Set Computer, procesory se složitější instrukční sadou
CRC - Cyclic Redundancy Check, je hašovací funkce pro detekci chyb
DMA - Direct Memory Access, technika přímého přístupu do paměti s minimální účastí procesoru
DOM - Document Object Model, objektově orientovaná reprezentace dokumentu
FCS - Frame Check Sequence, kontrolní byty na konci rámce
GNU GPL - GNU General Public License, všeobecná veřejná licence GNU
GNU LGPL - GNU Lesser General Public License, všeobecná veřejná licence GNU
GSM - Groupe Spécial Mobile, standard pro mobilní telefony
HTTP - Hypertext Transfer Protocol, internetový protokol pro výměnu dokumentů
JSON - JavaScript Object Notation, odlehčený formát pro výměnu dat
JTAG - Joint Test Action Group, architektura k testování a programování
MAC - Media Access Control adresa, jedinečný identifikátor síťového zařízení
MII - Media Independent Interface, standardní interface pro připojení Fast Ethernetu
NSU - Network system protection, Síťová ochrana
PHY - zkratka pro fyzickou vrstvu OSI modelu
RISC - Reduced Instruction Set Computers, procesory s omezenou instrukční sadou
RMII - Reduced Media Independent Interface, standardní interface pro připojení Ethernetu s redukovanými piny
SIM - Subscriber Identity Module, nese identifikaci účastníka v mobilní síti
SMS - Short Message Service, služba krátkých textových zpráv
SNTP - Simple Network Time Protocol, protokol pro synchronizaci vnitřních hodin
TCP - internetový protokol pro výměnu dokumentů
TCP/IP - Sadu protokolů pro komunikaci v počítačové síti
UDP - User Datagram Protocol, protokol pro přenos datagramů v počitačových sítí
URI - Uniform Resource Identifier, přesný identifikátor zdroje

Obsah

Přílohy

A. CD/DVD

1. Úvod

Na světě je mnoho automatizovaných úloh, od velkých robotů pro výrobu aut až po domácí pračky. Spousta věci, které by se dělaly celý den, uřídí procesor o velikosti lidského nehtu za pár mikrosekund. Stále se vyvíjí nové, rychlejší a dokonalejší zařízení a člověk jich má tolik, že je nestíhá všechny kontrolovat. Ten kdo vyvíjel tyto systémy nebyl bez chyby a nemohl očekávat všechny možné stavy a okolnosti při reálném použití. Proto je důležité, aby byly tyto systémy stále někým kontrolovány. Ovšem kontrolor zase chce, aby těch systému nemusel kontrolovat příliš hodně a příliš často a také, aby se nemusel měnit pozici, kde právě je.

Volnost přináší Internet. Notebooky, kiosky, mobily nebo i hodinky, ve všem dnes může být Internet. Informací je tam spousta, těch důležitých i těch zbytečných. Proč tedy nemít přístup k našich ~~chybovým~~ systémům přes něj? A i když se najde místo, kde Internet není, tak pravděpodobně tam je GSM[1] signál, přes který může být kdokoli obeznámen jedinou SMS[2] zprávou o případné chybě.

Úlohou této práce je vymyslet kontrolní systém (v tomto případě firmware), který oznámí uživateli případné problémy na jiných systémech nebo se mu dokonce zobrazí, co jiné systémy dělají. Připojením do GSM sítě a Internetu získá přístup k uživatelům. Webovou aplikací může zobrazit dostatek dat oprávněné osobě a v případě potřeby se může identifikovat problém.

Systém byl od začátku směrován pro sbírání různých informací ze systémů vyrábějící elektrickou energii. Tyto systémy jsou zapojeny do elektrické sítě přes zařízení s názvem Síťová ochrana. Toto zařízení sbírá i údaje a můj firmware je má převzít a zobrazit přes webové rozhraní. Vyvinutý systém má byt natolik univerzální, že přidání nového hlídaného zařízení do zdrojového kódu by neměl představovat problém.

Projekt byl pokračování mé pracovní činnosti, proto vše okolo hardwaru už bylo zadáno. Zvolena byla jedna z nejpoužívanějších architektur dneška. Protože nebyla nijak očekávaná vytíženost vstupními daty, je výkon zařízení, na kterém firmware běží, velmi malý a paměťové místo je dost omezené. To přinášelo nemalé problémy jak výběrem knihoven a operačního systému, tak i návrhem softwaru.

Celý systém jsem pojmenoval *Control Board Server* (CBS), protože zařízení, na které systém běží, se nazývá *Control Board* a protože hlavní úlohou mého systému je vytvářet webový *Server*.

Nakonec jen uvedu, že systém je stále vyvíjen, stále se do něj přidávají nové funkce a stále se dá zdokonalovat a opravovat. Proto se některé částí implementace zdají být dodělanější a některé méně.

První částí této práce budou předvedeny, případně vysvětleny použité technologie. V návrhové a implementační části budou popsaný všechny části vyvinutého softwaru. Implementace se snaží splnit zadané body této práce. V závěrečné části bude pojednáno o testování a zjištěných chybách.

1 Groupe Spécial Mobile, standard pro mobilní telefony.
2 Short Message Service, služba krátkých textových zpráv.

2. Technologie

2.1. Jádro systému

2.1.1. Architektura ARM

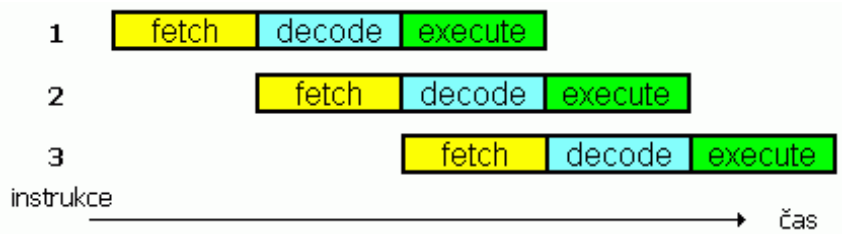
ARM je 32 bitová mikroprocesorová RISC[3] architektura tvořící jádro procesorů. Vyvinuta byla firmou ARM Limited a díky svým přednostem, jako je jednoduchý instrukční soubor a nízká spotřeba při vyšších frekvencích, se dnes používá na mnoha dnešních systémech a to od ovládačů výtahů po telefony, notebooky, herní konzole, přehrávače nebo i servery.

Protože ARM Limited už dnes nevyrábí ARM procesory, ale pouze vyvíjí její technologii, poskytuje ostatním firmám licence a schémata k výrobě. Vývojář má tak na výběr z mnoha výrobku od různých firem a může si vybrat přesně ten produkt, který mu vyhovuje. Vzhledem ke stejnému jádru u těchto výrobců jsou mikroprocesory ARM stejných generací a řad navzájem kompatibilní a kódy lze přenést z jednoho mikroprocesoru do jiného pouze s minimálními úpravami.

ARM mikrokontroléry jsou často vybaveny velkým množstvím modulů už od výroby, které přidávají další možnosti použití. Vývojářům také usnadňuje velké množství ARM kitů na trhu, které jsou také podpořeny množstvím vývojových prostředí.

Generace ARM7-TDMI

Jednou z novinek generace ARM7-TDMI se staly dvě instrukční sady, ARM a Thumb. Jejich volbou může vývojář zvolit mezi výkonem nebo šetřením paměti. Tato generace jako mnoho předešlých je založena na Von Neumanovém konceptu a používá tři stupňové zpracování (Obr. 1), a to *fetch* (načtení operačního kódu), *decode* (dekódování a příprava operandů) a *execute* (vykonání instrukce a zpětný zápis).



Obr. 1 Tři stupně zpracování instrukcí

Procesory z této generace zvládnou byty (8 bitů), půl-slova (16 bitů) a slova (32 bitů), při čemž půl-slova musí být v paměti zarovnané k dvou-bytové hranici a slova k čtyř-bytové hranici.

3 Reduced Instruction Set Computers, procesory s omezenou instrukční sadou.

Další vlastností je sedm operačních módů, do kterých se procesor může přepnout a ve kterých se používá různý počet registrů. Základní popis módů:

- *User:* Normální uživatelský ARM mód pro vykonávání instrukcí.
- *FIQ:* Designovaný pro podporu vysokorychlostního přenosu nebo kanálové procesy.
- *IRQ:* Používáno během řešení přerušení.
- *Supervisor:* Ochranný mód pro operační systém.
- *Abort mode:* Pro implementaci virtuální paměti a paměťové ochrany.
- *System:* Privilegovaný uživatelský mód pro operační systém.
- *Undefined:* Podpora softwarové emulace hardwarového koprocesoru.

Módy jsou pod softwarovou kontrolou nebo můžou být vyvolány vnějším přerušením. Většina aplikaci běží v základním uživatelským módu, ostatní slouží pro řešení přerušení nebo pro privilegované přístupy.

Následující popis instrukčních sad byl inspirovaný online seriálem [1].

Instrukční sada v režimu ARM

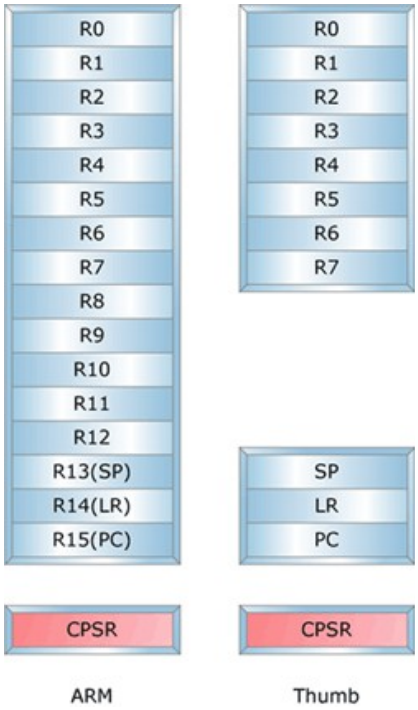
Mikroprocesory ARM byly zpočátku vybaveny jedinou instrukční sadou, v níž se nacházely instrukce o konstantní šířce 32 bitů. Vzhledem k tomu, že šířka externí datové sběrnice byla rovna taktéž 32 bitům a instrukce musely být zarovnané na celá slova, znamenalo to, že se celá instrukce vždy načetla jedinou operací, což je velký rozdíl oproti typickým mikroprocesorům s architekturou CISC[4], u nichž je délka instrukcí proměnná a mnohdy může přesahovat hranici slov. Tuto instrukční sadu lze na procesorech ARM používat dodnes a její největší předností je možnost uvést u každé instrukce podmínku, při jejímž splnění se má instrukce provést. Díky tomuto řešení je možné eliminovat velké množství skoků, jejichž provedení je samozřejmě problematické, a to nejenom na architektuře RISC, ale i na procesorech CISC.

Instrukční sada v režimu Thumb

I přes mnohé přednosti 32-bitové instrukční sady ARM se v některých případech objevují její nevýhody. Jedná se především o to, že použití 32-bitových instrukcí může mít menší hustotu kódu, což se projevuje větší délkou binárních souborů, větší pravděpodobností výpadku stránky z vyrovnávací paměti a taktéž vyšší cenou za zařízení (paměti). Z tohoto důvodu jsou procesory ARM patřící do novějších rodin vybaveny navíc další instrukční sadou pojmenovanou Thumb. Jedná se o instrukční sadu obsahující podmnožinu instrukcí vybranou na základě analýzy strojových programů generovaných překladači programovacích jazyků C a C++. Dále se v této instrukční sadě neobjevují bity určené pro podmíněné provádění instrukcí, což znamená, že je nutné se vrátit k použití podmíněných skoků. Na druhou stranu se však délka všech instrukcí zkrátila na šestnáct bitů, což

4 Complex Instruction Set Computer, procesory se složitější instrukční sadou.

dovoluje dosažení větší hustoty kódu. Nevýhodou může být nižší rychlost a nahrazování některých ARM instrukcí větším počtem Thumb instrukcí. Vzhledem k jednoduché implementaci Thumb do architektury (záležitost jednoho dekodéru) je možné s využitím speciální instrukce skoku přepínat mezi instrukční sadou Thumb a původní instrukční sadou ARM, a to dokonce i v rámci jednotlivých funkcí. Programátor či překladač tedy může využívat předností obou instrukčních sad. Posledním rozdílem je počet registrů, které instrukce mohou používat (Obr. 2).



Obr. 2 Rozdíl ARM a Thumb

I když není v generaci ARM7-TDMI, existuje i instrukční sada Thumb-2, která spojuje přednosti ARM i Thumb instrukčních sad. Instrukční sada Thumb-2 stále dosahuje vysoké hustoty kódu a to i při srovnatelném výkonu s ARM instrukcemi (přibližně 98%).

2.1.2. Mikrokontrolér Atmel *SAM7X/XC*

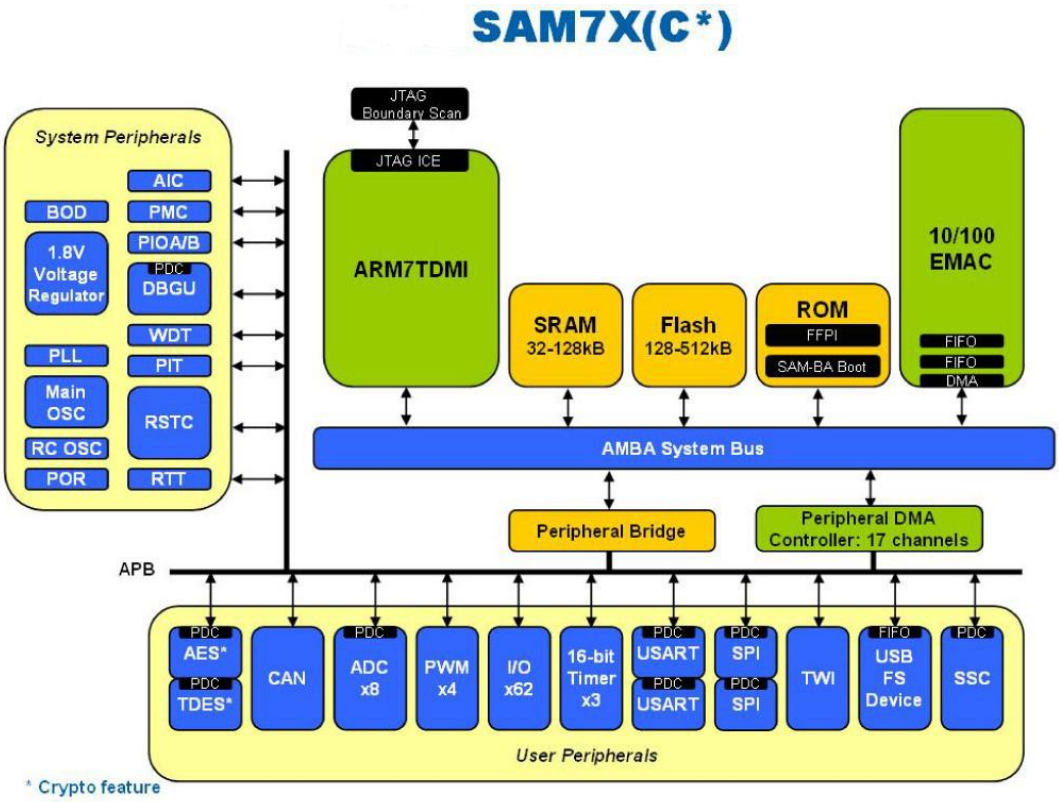
AT91SAM7X(C) je 32 bitový mikroprocesor s jádrem ARM7-TDMI vyráběný firmou Atmel (bližší popis v datasheetu [2]). Řada SAM7X se vyrábí v 6 variantách a to buď s kryptovacím AES modulem nebo bez něj a každá z těchto variant může mít 128kB, 256kB nebo 512kB flash paměti. SRAM paměti je 32kB, 64kB nebo 128kB podle zvoleného typu. Až na tyto vlastnosti je celá řada identická.

Základní specifikace AT91SAM7X:

- Založený na ARM7TDMI® ARM® Thumb® procesoru
- Vnitřní vysokorychlostní flash paměť (512KB, 256KB, 128KB)

- Optimalizační buffery pro běh Thumb instrukcí na maximální rychlost procesoru
- Vnitřní vysokorychlostní SRAM paměť (128kB, 64kB, 32kB)
- Maximálně 55 MHz při 1.65V a 85°C
- Systémový kontroler: Memory Controller (MC), Reset Controller (RSTC), Clock Generator (CKGR), Power Management Controller (PMC), Advanced Interrupt Controller (AIC), Debug Unit (DBGU), Periodic Interval Timer (PIT), Windowed Watchdog (WDT), Real-time Timer (RTT), 2x Parallel Input/Output Controllers (PIO), Peripheral DMA Controller (PDC)
- Periférie: USB 2.0 Full Speed Device Port (UDP), Ethernet MAC 10/100 base-T (EMAC), Part 2.0A and Part 2.0B Compliant CAN Controller (CAN), Synchronous Serial Controller (SSC), 2x Universal Synchronous/Asynchronous Receiver Transmitters (USART), 2x Master/Slave Serial Peripheral Interfaces (SPI), Three-channel 16-bit Timer/Counter (TC), Four-channel 16-bit Power Width Modulation Controller (PWMC), Two-wire Interface (TWI), 8-channel 10-bit Analog-to-Digital Converter (ADC)
- IEEE 1149.1 JTAG skenováni na všech digitálních pinech

Rozložení periferii v AT91SAM7X(C) lze shlédnout na následujícím obrázku (Obr. 3).



Obr. 3 Vnitřní diagram AT91SAM7X(C)

Popis důležitých periferii (použitých)

Memory Controller (MC)

Spravuje ASB sběrnici a kontroluje přístupy k ní, typicky od procesoru nebo od DMA[5]. Obsahuje adresní dekodér, správce stavů, detektor nezarovnání bytů a Embedded Flash Controller (EFC). EFC se stará o zápis na interní flash paměť jak softwarově tak i pomocí JTAG[6] zařízení.

Reset Controller (RSTC)

Stará se o reset procesoru nebo periférií. Resetuje i vnější zařízení pokud je připojeno na NRST pin. Zaznamenává poslední důvod resetu. Při zjištění nízkého napájení se postará o reset procesoru, aby nedošlo k nepředpokládaným výstupům.

Advanced Interrupt Controller (AIC)

Kontrolér, který spravuje přerušení od 32 zdrojů. Má 8 stupňů priorit, při čemž vyšší priorita může přerušit nižší. Priority lze také nastavit na různé druhy citlivosti.

Periodic Interval Timer (PIT)

Poskytuje operačnímu systému přerušení pro plánovač. Garantuje maximální přesnost a efektivní řízení.

Real-time Timer (RTT)

32-bitový čítač určený pro počítání uplynulé sekundy. Generuje periodicky přerušení a automaticky se nastavuje na požadovanou hodnotu.

Parallel Input/Output Controllers (PIO)

Spravuje 32 plně programovatelných vstupně/výstupních pinů. Každý z těchto pinů může sloužit k univerzálním I/O operacím nebo může být přiřazen k příslušným zabudovaným perifériím. Také každý z těchto pinů zvládne přerušení s možnosti filtrace a řízení pull-up rezistoru.

Peripheral DMA Controller (PDC)

Stará se o přenos dat mezi danými periferiemi a pamětí s minimální účastí procesoru (DMA). Tím je omezen velký počet přerušení hlavně u periferii s velkým tokem dat (EMAC, UDP) . Přináší i úsporu energie a jednodušší ovládání periferií. Spravuje 13 kanálů (další 4 pokud je přítomný šifrovací modul), a to dva páry pro USART, dva páry pro SPI, pár pro SSC a DBGU, zbylý třináctý kanál pro ADC.

5 Direct Memory Access, technika přímého přístupu do paměti s minimální účastí procesoru
6 Joint Test Action Group, architektura k testování a programování.

Ethernet MAC 10/100 base-T (EMAC)

Implementuje 10/100 Ethernet MAC kompatibilní se standardem IEEE 802.3. Kontroluje adresy příchozích rámců a to i pro multicast. Obsahuje i základní statistiku počtu příchozích rámců.

Universal Synchronous/Asynchronous Receiver Transmitters (USART)

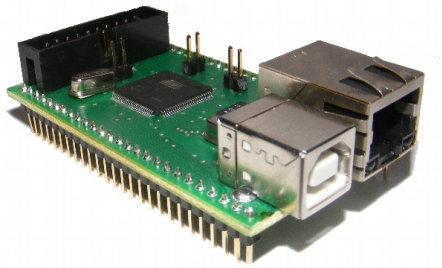
Poskytuje plně duplexní univerzální synchronní/asynchronní sériovou linky a také velkou škálu nastavení (délka dat, parita, počet stop bitů) s množstvím podporovaných standardů. Dokáže detekovat velké množství chyb na sběrnici (např chyby přerušení nebo přetečení bufferů).

Master/Slave Serial Peripheral Interfaces (SPI)

Synchronní sériová linka která komunikuje s externími zařazeními v Master nebo Slave módu. Dokáže komunikovat s externím procesorem pokud je připojen ke sběrnici.

2.1.3. Kit Kramara AT91SAM7X512-KIT

Vývojový kit AT91SAM7X512-KIT (Obr. 4) je postaven na mikrokontroléru AT91SAM7X512. Na kitu je datová paměť Flash AT45DB041D o kapacitě 4Mbit připojená k pinům SPI mikrokontroléru. Pro připojení Ethernetu je kit vybaven PHY[7] čipem DAVICON Semiconductor DM9161A a k němu připojeným konektor RJ-45. Konektor JTAG na desce slouží k ladění a nahrávání softwaru. Mikrokontrolér je taktován hodinovým kmitočet 18,432MHz (pomocí krystalu). Díky rastrování konektorů desky (2,54mm) lze kit využívat na nepájivém poli pro pohodlné vyvíjení. Kit přináší ověřenou konstrukci a jednoduchou montáž a implementaci.



Obr. 4 AT91SAM7X512-KIT

2.2. Připojené komponenty a komunikační protokoly

2.2.1. Modem GSM -Teltonica ModemCOM/G10

Modem GSM je specializovaný typ modemu, který po přijetí SIM[8] karty a ověření operátorem dokáže posílat data přes GSM síť. Jsou rozšířené jak pro osobní tak průmyslové použití a lze se přes ně i připojit na Internet nebo posílat SMS zprávy.

ModemCOM/G10 (Obr. 5, přejmenováno na WirelessCOM) je GSM modem vyrobený italskou

7 Zkratka pro fyzickou vrstvu OSI modelu
8 Subscriber Identity Module, nese identifikaci účastníka v mobilní síti.

firmou Teltonica a je založen na GPRS modulu Teltonika TM2.

Základní specifikace modemu:

- Možnost připojení k internetu: GPRS class 10 (až 56 – 114 kbps), CSD (až 14,4 kbps)
- Quad-Band GSM 850 / 900 / 1800 / 1900 MHz
- Funkce: TCP/IP Stack, uložení a čtení Flash, SMS (text/data)
- Interní nebo externí GSM anténa
- Konektory: DB9 female, konektor zdroje, slot pro SIM kartu, SMA konektor pro GSM anténu

Obr. 5 ModemCOM/G10

AT příkazy

Tyto příkazy slouží ke komunikaci s GSM modemem. Pro různé typy modemů mohou být různé tvary příkazů, ale ty základní jsou většinou pro všechny stejné. Následující vysvětlení příkazů bude pro zmiňovaný modem Teltonica ModemCOM/G10, na jiných modemech nejsou otestované. Modem má velké možností nastavení, ve kterých lze změnit i způsob zadávání a vrácení příkazu, proto se počítá s výchozím nastavením (od výroby).

AT příkaz může mí 3 základní tvary:

- Existence příkazu: AT+<příkaz>=?<CR>[9]
- Načtení hodnot: AT+<příkaz>?<CR>
- Zápis hodnot: AT+<příkaz>=<parametr><CR>
- Pouze příkaz: AT+<příkaz><CR>

Každý příkaz nemusí nebo může mít několik parametrů oddělený čárkou. Příkaz musí být ukončený znakem <CR>. Odpovědí modemu je vždy vrácený zadaný příkaz (pro kontrolu) a řetězec OK<CR><LF>[10] pro přijatý příkaz nebo ERROR<CR><LF> pro chybu při zpracovávání

9 Carriage Return, v ASCII hodnota 0x0d.
10 Line Feed, v ASCII hodnota 0x0a.

příkazu. Znaky nerozeznané jako příkaz jsou ignorovány.

Příklady příkazu (kontrola komunikace a poslání SMS):

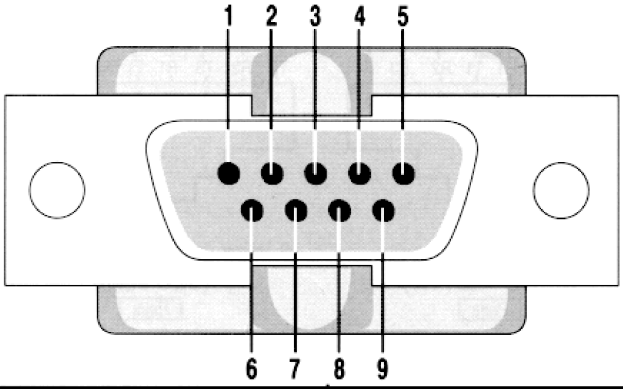
AT příkaz	Příklad příkazu	Příklad odpovědi	Popis
AT	AT<CR>	OK<CR><LF>	Testuje spojení s modemem.
AT+CMGF	AT+CMGF=1<CR>	OK<CR><LF>	Změní režim na textový (z PDU).
AT+CMGS	AT+CMGS="+420987654321"<CR> "text SMS"<Ctrl+z>[11]	OK<CR><LF>	Posílá SMS na dané číslo (textový režim).

Standard RS-232 a hardwarový handshaking

Standard RS-232 se používá jako sériové komunikační rozhraní mezi zařazeními na delší vzdálenost (20m). Dnes už je nahrazováno modernějšími standarty jako je USB, ale v průmyslu je stále používán pro svou odolnost proti rušení při nízkých rychlostech a jednoduchost.

Standard definuje asynchronní sériovou komunikaci pro přenos dat. Pořadí přenesených datových bitů je od nejméně významného bitu po bit nejvýznamnější. Počet přenesených dat se pohybuje od 7 po 9 bitů. Logická 0 a 1 je reprezentována pomocí dvou úrovní napětí (od ±5V po ±15V) a je převáděná z a do TTL logiky například pomocí čipu MAX232 od firmy Maxim.

Standard používá jako konektory například některý z D-Sub (DB-9, DB-25) nebo i RJ-45. Na následujícím diagramů (Obr. 6) je popis DB-9 konektoru, který je použit při připojení modemu Teltonica ModemCOM/G10. Piny TxD (Transmitted Data) a RxD (Received Data) k odesílání a příjmu dat, ostatní piny slouží k řízení nebo k signalizaci.



Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

Obr. 6 Standard RS-232 na konektoru DB-9

11 Stisknutí klávesy, nahrazeno ASCII hodnotou 0x1a.

Hardwarový handshaking je jeden z druhu řízení toku dat. Definuje připravenost k přenosu a jeho zahájení na úrovni hardwarového rozhraní. Tento druh používá piny RTS (Request To Send)/CTS (Clear To Send) nebo DTR (Data Terminal Ready)/DSR (Data Set Ready) k oznámení druhé straně o připravenosti přijímat data. Modem Teltonica ModemCOM/G10 používá ve výchozím nastavení řízení toku pomocí RTS/CTS a pro připojení se používá přímý kabel.

2.2.2. Síťová ochrana UNIMA-KS

Síťová ochrana (NSU[12], Obr. 7) slouží pro ochranu sítě před nežádoucími vlivy zdrojů energie připojených k síti. Může taky sloužit na ochranu zařízení připojených k síti, které jsou citlivé na změnu stavu napájecí sítě. Vyhodnocuje nejen napětí, frekvenci a fázi jednotlivých napětí, ale také měří proud, činný a jalový výkon (dle varianty provedení 3 nebo 6-fázově). K zařízení lze připojit pomoci standartu RS-485 další zařízení, které bude moci sbírat data ze síťově ochrany. Díky konfiguračním možnostem je síťová ochrana vhodná například pro použití v solárních elektrárnách.

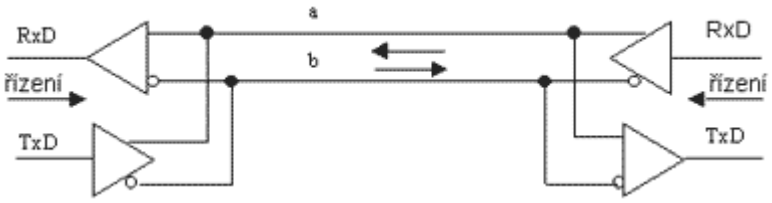
Obr. 7 UNIMA-KS Síťová ochrana (verze 2)

Standard RS-485

RS-485 (nebo EIA-485) je standard sériové komunikace používaný především v průmyslovém prostředí. Standard je navržen tak, aby umožňoval vytvoření dvouvodičového poloduplexního vícebodového sériového spoje. Má stejný základ jako standard RS232, od kterého se liší především jinou definicí napěťových úrovní, nepřítomností modemových signálů, možností vytváření sítí (též sběrnice) sestávající z až 32 zařízení a možností komunikace na vzdálenost až 1200 metrů. Výhodou rovněž je, že linku RS-485 je možné vytvořit z široce rozšířeného standardu RS-232 pomocí jednoduchých převodníků úrovně.

Logické úrovně se určují jako rozdíl na dvou linkách (Obr. 8), kterými jsou zařízení propojena (vyskytuje se i linka GND). Tento režim má pouze poloviční duplex, proto se mohou objevit i další dvě linky pro plný duplex. Linky se nazývají A a B. Pro logickou 1 musí být rozdíl A - B < -200 mV a pro logickou 0 musí být A - B > +200 mV.

12 Network system protection.



Obr. 8 Standard RS-485 na dvou linkách

Komunikační protokol Modbus

Modbus je otevřený protokol pro vzájemnou komunikaci různých zařízení, který umožňuje přenášet data po různých sítích a sběrnicích (RS-232, radiový přenos, ...). Komunikace funguje na principu předávání datových zpráv mezi klientem a serverem (master a slave). Protokol definuje svou strukturu zpráv, při čemž existují dva typy:

- RTU režim, při kterém je byte vysílán jako jeden znak a na konci vysílání se přidává kontrolní součet CRC[13]. Vysílání musí byt souvislé (mezera mezi znaky není větší než 1,5 znaku) a mezi zprávami musí být pomlka větší jak 3,5 znaku. Tento režim se používá nejčastěji.
- ASCII režim, při kterém je byte vysílán jako dva znaky a zpráva začíná znakem : a končí znaky <CR><LF>. Mezi znaky může být větší mezera.

Protokol definuje i adresaci. Adresa 0 je pro všesměrové vysílání a adresy od 1 do 247 jsou adresy připojených zařízení. Protokol také definuje různé druhy funkcí (žádostí na zařízení), v případě Síťové ochrany je jen implementována funkce 4 („Read Input Registers“).

Příklad komunikace u síťové ochrany (dotaz a odpověď):

Adresa	1 byte	0x00 - 0xff
Kód funkce	1 byte	0x04
Adresa první registru	2 byty	0x0000 - 0x0011
Počet registrů ke čtení (N)	2 byty	0x0001 - 0x0011
Kontrolní součet	2 byty	CRC16

Adresa	1 byte	0x00 - 0xff
Kód funkce	1 byte	0x04
Počet datových bytů	1 byte	2 * N
Hodnoty čtených registrů	2 * N bytů	
Kontrolní součet	2 byty	CRC16

13 Cyclic Redundancy Check, je hašovací funkce pro detekci chyb

2.3. Síťové protokoly

2.3.1. Protokol SNTP

SNTP[14] je protokol pro synchronizaci vnitřních hodin zařízení s proměnným zpožděním. Tento protokol zajišťuje, aby všechny zařízení v počítačových sítích měly stejný a přesný čas. Byl obzvláště navržen tak, aby odolával následku proměnlivého zpoždění v doručování paketů. Pracuje nad protokolem UDP[15].

SNTP klient používá Marzullův algoritmus pro stanovení času z (nepatrně) se lišících odpovědí časových serverů. Používá se čas UTC se speciálními příznaky pro přestupné sekundy. Běžně se s algoritmem dosahuje přesnosti hodin v řádu milisekund.

Algoritmus:

```
d = (T4 - T1) - (T2 - T3)      t = ((T2 - T1) + (T3 - T4)) / 2.

d – časové zpoždění při žádosti
t – časový posun od nynějšího času na zařízení
T1 – čas žádosti poslaný klientem
T2 – čas žádosti přijatý serverem
T3 – čas odpovědí poslány serverem
T4 – čas odpovědí přijatý klientem
```

NTP zpráva posílaná na server se skládá minimálně z 48 bytů. Pro jednoduchého klienta je většina části zprávy nepovinná. Časy jsou zadávány v 64-bitových hodnotách, kde vrchní 32-bitová část jsou sekundy a zbylých spodních 32 bitů je část sekundy. Hodnoty jsou v zarovnání Big-Endian[16]. Pro jednoduché použití stačí vyplnit *T3* časem kdy se zpráva odesílá a přijmout čas *T2*. *T1* je čas klienta před posláním a *T4* po poslání. Po použití algoritmu je *t* posun v čase od aktuálního na klientovi.

2.3.2. Protokol HTTP

HTTP[17] je internetový protokol pro výměnu dokumentů pracujícím nad protokolem TCP[18]. Pro svou jednoduchost se stal jedním z nejpožívanějším protokolem na Internetu. Bylo vydáno několik verzí protokolu, proto zde bude popsána jen poslední a nejpoužívanější verze 1.1.

Každá zpráva se vždy skládá z jednoho inicializačního řádku, z několika hlaviček a z těla zprávy. První řádek a hlavičky jsou ukončené znakem <LF> nebo <CR><LF> (preferované). Mezi hlavičkami a tělem zprávy (i když žádné tělo zpráva neobsahuje) je prázdný řádek pomocí <CR><LF>. Tvar inicializační hlavička je dán typem zprávy.

14 Simple Network Time Protocol.
15 User Datagram Protocol, protokol pro přenos datagramů v počitačových sítí.
16 Na nejnižší adresu se uloží nejvýznamější byte.
17 Hypertext Transfer Protocol.
18 Transmission Control Protocol, protokol pro přenos segmentů v počitačových sítí.

Pokud je zpráva žádosti, skládá se ze tří částí oddělené mezerou: typu zprávy, URI[19] a verze protokolu HTTP. Typů zpráv existuje několik, nejhlavnějšími jsou GET, POST a HEAD. Ostatní typy nemusí být implementovány. HEAD je to samé jako GET jen mu chybí tělo zprávy a POST má jiný způsob implementace v prohlížečích než GET (může v žádosti obsahovat tělo zprávy, neukládá se do vyrovnávací paměti). URI obsahuje relativní adresu ke zdroji dat na serveru. Může také obsahovat dodatečné data napsané za adresou oddělené otázníkem (u typu GET nebo HEAD).

Pokud je zpráva odpovědí, skládá se ze tří částí oddělené mezerou: verze protokolu HTTP, číslo odpovědi a text odpovědi (dle čísla odpovědi). Číslo a text odpovědi jsou dané podle informace, kterou zpráva nese. Například pokud žádost GET je v pořádku a zdroj dat je nalezen, odpověď bude s číslem 200 a s textem OK.

Hlavičky přidávají dodatečné informace o zprávě. Může nést například informaci o jazyku, kódování, autentizaci nebo typu obsahu zprávy (tzv. MIME). Verze HTTP 1.1 má 46 možných hlaviček a některé jsou povinné (Host, The Date, ...).

Příklad žádosti:

```
GET /soubor.html HTTP/1.1<CR><LF>
Host: www.host.com:80<CR><LF>
From: user@client.com<CR><LF>
User-Agent: HTTPTool/1.1<CR><LF>
<CR><LF>
```

Příklad odpovědi:

```
HTTP/1.0 200 OK<CR><LF>
Date: Fri, 31 Dec 1999 23:59:59 GMT<CR><LF>
Content-Type: text/html<CR><LF>
Content-Length: 999<CR><LF>
<CR><LF>
<html>
  .
  .
  .
</html>
```

Pokud jsou data posílané v žádosti ve formě textu jsou některé znaky zabaleny do číselné podoby dané znakové sady. Dvojčíslí za znakem % značí číslo znaku v hexa ze znakové sady (pro jeden znak může být více znaku % a dvojčíslí za sebou). Pro mezeru, která má častý výskyt, se může objevit jako náhradní znak +.

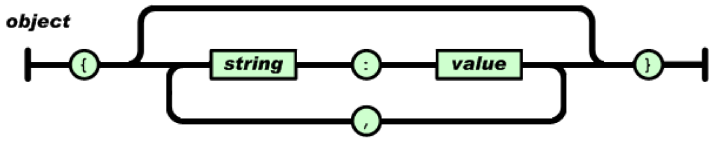
Verze 1.1 zvládá jednoduchou autentizaci. Pokud na nějakou žádost odpoví server číslem 401 a hlavička bude obsahovat typ požadované autentizace, tak klient zopakuje žádost z daným jménem a heslem a pokusí se o přístup. Jestliže autentizace projde, bude do každé další zprávy přidávat hlavičku s přihlašovacími údaji. Tento způsob autentizace ovšem není bezpečný a postrádá jakékoli šifrování (pouze se skrývá za algoritmus).

19 Uniform Resource Identifier, přesný identifikátor zdroje.

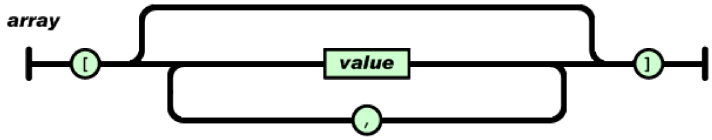
2.3.3. Komunikační formát JSON a technologie AJAX

JSON[20] je odlehčený formát pro výměnu dat. Je jednoduše čitelný i zapisovatelný člověkem a snadno analyzovatelný i generovatelný strojově. Je založen na podmnožině programovacího jazyka JavaScript, proto ho lze v něm bez složitých konverzí i použit. JSON je textový, na jazyce zcela nezávislý formát.

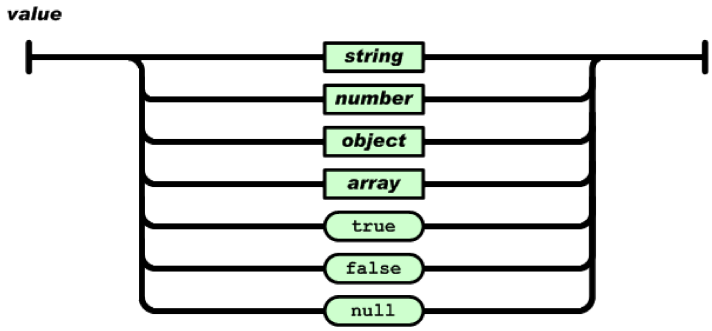
V JSON jsou realizovány datové konstrukce objekt (Obr. 9), pole (Obr. 10), hodnota (Obr. 11), řetězec (Obr. 12) a číslo (Obr. 13). Obrázky jsou převzaty ze stránek [3], kde se nachází i bližší popis.



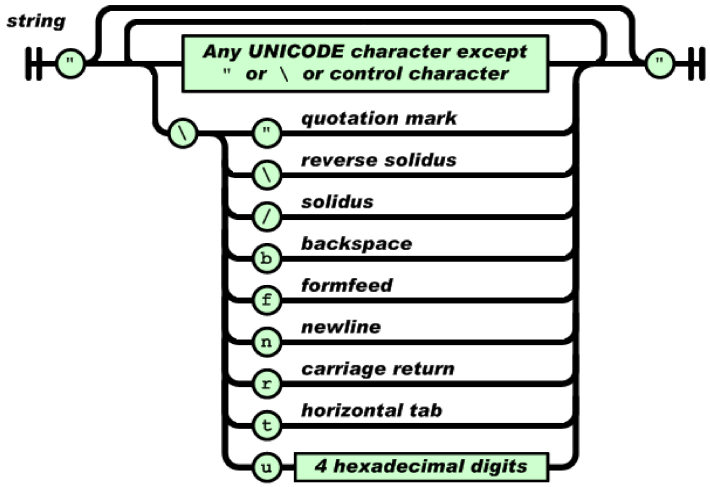
Obr. 9 JSON - objekt



Obr. 10 JSON - pole

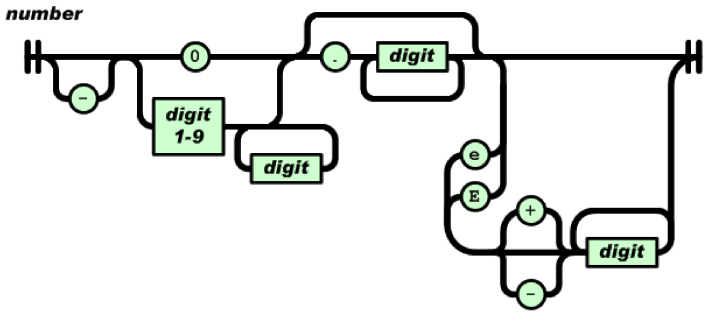


Obr. 11 JSON - hodnota



Obr. 12 JSON - řetězec

20 JavaScript Object Notation.



Obr. 13 JSON - číslo

AJAX[21] je označení pro technologie vývoje interaktivních webových aplikací, které mění obsah svých stránek bez nutnosti jejich znovunačítání. Pro práci s AJAXem jsou nutné technologie DOM[22] a programovací jazyk JavaScript pro dynamickou změnu informací na straně klienta. Protože implementace se různí dle použité platformy, používají se knihovny. Technologie nezvyšuje nároky na schopnost použitých serverů, protože se posílají dotazy GET a POST jak při běžné HTTP komunikaci.

2.4. Programové knihovny

2.4.1. Knihovna FreeRTOS

FreeRTOS [4] je škálovatelný operační systém reálného času pro vestavné systémy napsáný v programovacím jazyce C. Všechny verze se vydávají pod licencí GNU GPL[23] s výjimkami obsažené na stránkách projektu. Podporuje jak plně preemptivní tak kooperativní zpracování vláken. Jádro tohoto RTOS je složeno pouze ze tří souborů, a to *task.c, queue.c* a *list.c*. Od verze 7.0.0 je také přidán softwarový časovač v souboru *timers.c*. Pro správnou funkci systému jsou potřeba i další zdrojové soubory, které jsou přibaleny ve vydání. Balík také obsahuje mnoho příkladů použití, ze kterých se dá vycházet při zažínání nového projektu.

Pro komunikaci mezi vlákny nebo mezi přerušeními a vlákny poskytuje operační systém komunikační a synchronizační nástroje typu fronta zpráv, semafor a mutex. Nástroj mutex zároveň řeší inverzi priorit. Ovšem v jádře systému existuje jen synchronizační nástroj fronta zpráv, zmiňovaný mutex a semafor jsou pouze nadstavbou a nenesou žádné urychlení při použití.

Hlavní výhodou tohoto operačního systému je jeho minimalistická implementace. Pokud nějakou část sytému vývojář nepotřebuje, může zvolit jeho vyňatí z kompilace. Tímto způsobem lze vytvořit na míru systém řešící daný problém s minimem spotřebované pamětí a procesorového času. Toto nastavení se provádí v hlavičce FreeRTOSConfig.h i s ostatním nastavením operačního systému (např. priority, velikost haldy, rychlost operačního tiku jádra).

Pro zabudovaní systému do daného mikrokontroléru je potřeba port. Protože FreeRTOS je napsán multiplatformně, tak port provede propojení mezi danou platformou a samotným systémem.

21 Asynchronous JavaScript and XML.
22 Document Object Model, objektově orientovaná reprezentace dokumentu.
23 GNU General Public License, všeobecná veřejná licence GNU.

Nejdůležitější požadovanou částí je zpřístupněný časovač a softwarové přerušení. FreeRTOS oficiálně podporuje 31 portů a mnoho dalších portů vytváří komunita.

Další důležitou části je řešení haldy pro dynamicky alokovanou paměť. FreeRTOS obsahuje 3 způsoby řešení haldy obsažené v souborech heap_1.c, heap_2.c a heap_3.c.

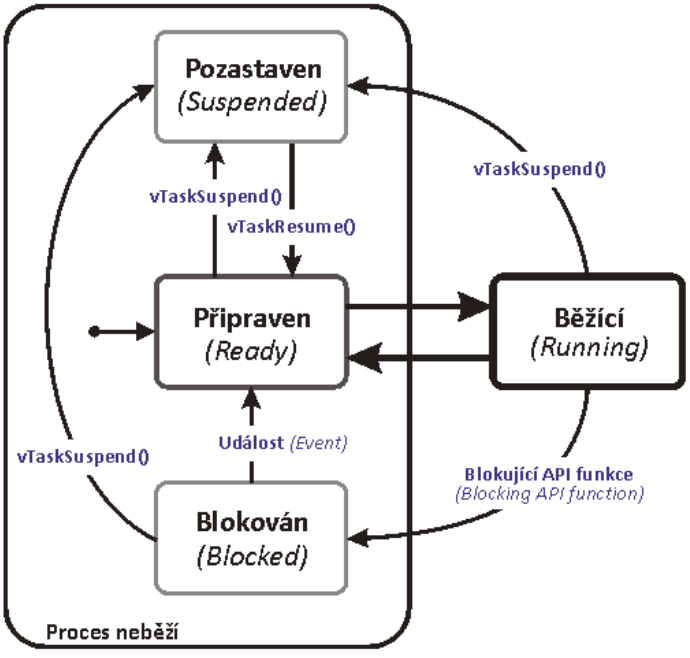
1. Jednoduchý systém alokace, ve kterém není povoleno uvolňovaní nepotřebné paměti. Výhodou je determinismus a jednoduchý kód.
2. Složitější systém alokace, který oproti první verzi dokáže dealokovat nepotřebnou paměť. Nevýhoda je ztráta determinismu a možná fragmentace.
3. Speciální verze používající funkce malloc() a free() ze systémových knihoven. Je to pouze zabalení funkcí pro použití na PC.

Následující popis částí je inspirovaný online článkem [5].

Plánovač

FreeRTOS disponuje plánovačem s prioritním plánováním, kdy je každému vláknu přiřazena hodnota jeho priority. Menší číslo znamená nižší přiřazenou prioritu, přičemž nejnižší možná hodnota je 0. V případě vláken se stejnou prioritou jsou vlákna plánována metodou kruhu (Round-robin). To znamená, že jsou pravidelně střídána. Maximální priorita je omezena na námi požadovanou hodnotu. Nultou prioritu má i vlákno nečinnosti, které lze zpřístupnit a použít například pro zapnuti různých úspor energie.

Na následujícím obrázku (Obr. 14) jsou znázorněny stavy, ve kterých se vlákno může nacházet.



Obr. 14 Stavy plánovače

V jednom okamžiku může být procesoru přiděleno pouze jedno vlákno, a to odpovídá stavu *Běžící*. Ostatní vlákna jsou v tom okamžiku v neběžícím stavu, který se pak dále rozděluje:

- *Pozastaven* - vlákna v tomto stavu nejsou přístupná plánovači. Každé vlákno lze umístit a vyjmout z tohoto stavu funkcemi *vTaskSuspend* a *vTaskResume*.
- *Připraven* - vlákna v tomto stavu jsou připravena ke zpracování a čekají na přidělení plánovače.
- *Blokován* - vlákna čekající na událost. Události mohou být časové (vlákno je zablokováno na určitý čas) a synchronizační (semafor, mutex, fronta zpráv).

Jelikož je FreeRTOS velice jednoduchý, neexistuje v něm striktní oddělení paměťového prostoru mezi vlákny. V systému se může vytvořit tolik vláken, kolik je operační paměti. Každé z nich si ukousne jeho část (zásobník, informace o vláknu) a s ní pak pracuje. Kolik paměti si vlákno vezme, se volí při vytváření, atributem funkce *vTaskCreate*. Mimo to že se vlákno může vytvořit, tak se může i smazat, změnit priorita nebo uspat.

Synchronizační nástroje

Při běžném programování mikroprocesorů bez operačního systému je snaha psát kód tak, aby se nikde zbytečně nezastavoval v čekacích smyčkách, čímž by se omezovala činnost ostatních částí programu. Naopak při psaní vláken s preemptivním plánováním je snaha využívat synchronizačních nástrojů, které zajistí čekání na vnější událost blokací vlákna.

Mnoho z následujících popsaných synchronizačních nástrojů obsahuje funkce pro použití v přerušení. Tyto zvláštní funkce jsou jednoduší a rychlejší, aby bylo přerušení co nejrychleji vyřízeno a ukončeno.

Mutex

Mutex je asi nejjednodušší synchronizační nástroj, který řeší úlohu takzvaného výlučného přístupu. Využívá se tam, kde může přistupovat do zařízení (paměti) vždy pouze jedno vlákno. Příkladem je přístup ke komunikačnímu rozhraní. Transakce čtení ze zařízení se skládá z několika kroků, které nesmí být ve svém průběhu přerušeny, a proto před samotným zahájením komunikace se provede pokus o získání (zamknutí) mutexu funkcí *xSemaphoreTake*. Pokud jiné vlákno v tomto okamžiku nekomunikuje (nevlastní mutex), mutex je získán prvním vláknem, zahájí se komunikace a po jejím skončení ho zase uvolní funkcí *xSemaphoreGive*. V případě, že jiné vlákno již mutex vlastní, operační systém zajistí, že prvnímu vláknu nebude přidělován čas procesoru do té doby, dokud nebude mutex opět volný (vlákno bude ve stavu *Blokován*).

Binární semafor

Binární semafor se využívá pro synchronizaci dvou vláken, nebo také hardwarového přerušení a vlákna. Využívají se především v úlohách typu producent/konzument. Hodnota binárního semaforu

nabývá nula a jedna. Funkcí *xSemaphoreGive* se jeho hodnota nastaví na jedna, i když už před tím tuto hodnotu měl (tato funkce nemá blokující charakter), a funkcí *xSemaphoreTake* se hodnota semaforu nastaví na nulu, ale jen v případě, že měl hodnotu jedna. Pokud se v semaforu již nula nachází, vlákno se přesune do blokovaného stavu blokován, a bude opět spuštěno až některé z vláken hodnotu semaforu nenastaví na jedna.

Semafor

Semafor je další nástroj pro synchronizaci vláken. Může být použit pro počítání libovolných událostí nebo opět pro úlohu typu producent/konzument. Je obdobou binárního semaforu, ale s tím rozdílem že jeho hodnota může nabývat kladných čísel včetně nuly. Pomocí funkcí *xSemaphoreGive* a *xSemaphoreTake* se inkrementuje a dekrementuje hodnota semaforu, podle kterého se řídí běh vláken.

Pokud se zavolá funkce *xSemaphoreTake* na semafor, který má hodnotu větší než 0, tak vykonávání programu nebude zablokováno. Pokud se zavolá funkce *xSemaphoreTake* na semafor, který má nulovou hodnotu, tak vykonávání programu bude blokováno do té doby, dokud jiné vlákno nezvýší semafor do kladných hodnot pomocí funkce *xSemaphoreGive*.

Fronta zpráv

Fronta zpráv je ideální v případě, kdy je potřeba nejenom dvě nebo více vláken synchronizovat, ale také rovnou mezi nimi i předávat data. Při pokusu o čtení prázdné paměti zařídí blokaci vlákna, které se o to pokusilo. Tím se zajistí, že vlákno, které slouží ke zpracování dat, ale nemá tyto data k dispozici, nedostává výpočetní čas procesoru do té doby, dokud nejsou nějaká data opět dostupná. Mezi dvě hlavní API funkce, pomocí kterých pracujeme s frontou zpráv, jsou *xQueueSend* a *xQueueReceive*.

Softwarový časovač

Nejnovějším přírůstkem v posledním vydání se stal vylepšený softwarový časovač. Ten přidává novou možnost volání úkolů po určitém čase, čímž odpadá použití přerušení. Výhodou je, že časovač nespotřebovává žádný procesorový čas navíc, dokud se nezavolá daná řešící funkce.

2.4.2. Knihovna lwIP

LwIP je odlehčená implementace TCP/IP[24] protokolu od Adama Dunkelse. Cílem této implementace je redukovat potřebnou paměťovou náročnost knihovny. Autor udává, že základní jádro spotřebuje přibližně 40 kB programové a 10 kB operační paměti. Proto je tato knihovna mířena na vestavěné nebo i deterministické systémy.

Od stejného autora pochází i projekt uIP, který je ještě více odlehčený. Ten vyžaduje k provozu pouze 5 kB programové a 2 kB operační paměti. Jeho výkon je ale velmi nízký, neobsahuje tolik vylepšení jako lwIP a nerozumí si s vlákny operačního systému na který je portován (vytváří si vlastní

24 Sadu protokolů pro komunikaci v počítačové síti.

pseudo vlákna). Největší nevýhodou je, že poslední aktualizace je z roku 2001 a dále není opravován a vyvíjen. Přesto se ale objevuje jako přídavek k různým operačním systémům a to většinou jeho upravená verze.

Knihovna lwIP je zdarma přístupná a napsaná v programovacím jazyce C. Od jejího vydání je velmi žádaná a nyní se používá v mnoha komerčních produktech. Je portovaná na mnoho platforem a může běžet bez kontroly a řízení operačního sytému. Pokud je je portována na operační systém, zvládne synchronizaci vláken.

Knihovna obsahuje následující implementaci protokolů a vylepšení:

- IP (Internet Protocol) včetně předávání paketů přes několik síťových interfaců
- ICMP (Internet Control Message Protocol) pro síťovou údržbu a ladění
- IGMP (Internet Group Management Protocol) pro multicastový management
- UDP (User Datagram Protocol) včetně experimentálního UDP-lite rozšíření
- TCP (Transmission Control Protocol) s kontrolou přetížení
- Nativní API přistupující přímo k jádru knihovny (nepoužívá synchronizaci pro vlákna)
- Netconn API jako odlehčené API se synchronizaci vláken (možnost bez-kopírovacího API při průchodu knihovnou)
- Soketové API podobné Berkeley soketům
- DNS (Domain names resolver)
- SNMP (Simple Network Management Protocol)
- DHCP (Dynamic Host Configuration Protocol)
- AUTOIP (shoduje se s RFC 3927) pro IPv4
- PPP (Point-to-Point Protocol)
- ARP (Address Resolution Protocol) pro Ethernet

2.4.3. Knihovna Prevel

Prevel je odlehčena (jádro přibližně 5kB) JavaScript knihovna, která usnadňuje používání multiplatformních funkcí a řeší některé základní problémy při tvorbě webových aplikací.

Knihovna obsahuje:

- Core: základní funkce pro potřebu knihovny, práce s JSON.
- Ajax: implementuje posílání a příjem zpráv a to metodou text nebo JSON (řeší multiplatformní rozdíly).

- Attr: vylepšená práce s elementy objektů.
- Css: vylepšený zápis stylů do objektů.
- Events: vylepšený zápis událostí.
- Find: vylepšuje hledání elementů v HTML.
- Insert: vytváří lepší zápis pro vkládání elementů a textů do HTML.
- Manipulate: manipuluje z elementy v HTML.
- Ostatní: další vylepšení, která nejsou součástí základní verze.

Knihovna je dodávaná v mnoha variantách. První možnost je stažení souborů s různými třídami funkcí zvlášť, v kterých se musí řešit závislosti. Další možnost je stažení jednoho souboru se všemi základními třídami. Poslední možnost je podobná druhé jen v ní nejsou komentáře a je napsána v minimalistické podobě (žádné odřádkování, omezení mezer) zabírající přibližně 16 kB.

2.5. Implementační nástroje

2.5.1. Vývojová platforma Eclipse

Projekt Eclipse byl zahájen ve společnosti IBM v roce 2001 a v současnosti se jedná o oblíbenou domácí i firemní vývojovou platformu. Jedná se o propracovanou a univerzální aplikaci, která se velmi často používá ve funkci integrovaného vývojového prostředí (IDE) a to především pro vývoj programů v Javě. Ve skutečnosti jsou možnosti Eclipse mnohem větší, protože díky svému systému přídavných modulů může velmi dobře posloužit i pro vývoj aplikací v programovacích jazycích C, C++, Python, PHP a dalších. Kromě toho existují pluginy pro tvorbu XML souborů, návrhu UML diagramů a pro správu webových aplikací.

Eclipse je postaveno na programovacím jazyku Java, čímž je zaručena jeho poměrně snadná přenositelnost na různé platformy. Samotné jádro projektu Eclipse je relativně malé, zejména v porovnání s některými „monolitickými“ vývojovými prostředími. Díky koncepci rozšiřujících modulů je možné do Eclipse přidávat další funkce – nové typy editorů, podporu pro další programovací jazyky, ladicí nástroje, profilery, návrháře grafického uživatelského rozhraní nebo napojení na aplikační servery.

2.5.2. Vývojové nástroj YAGARTO

YAGARTO je vývojové prostředí pro platformu ARM běžícím na operačním systému Microsoft Windows nebo Mac OS X. Je vydáno pod licencí GNU GPL nebo GNU LGPL[25]. Je v něm vložený GNU C/C++ toolchain, který obsahuje nástroje pro tvorbu a ladění aplikací.

25 GNU Lesser General Public License, všeobecná veřejná licence GNU.

Obsah projektu:

- Binutils: Soubor nástrojů používaných při vývoji softwaru, který slouží pro manipulaci s objektovým kódem v různých formátech.
- Newlib: Jedna z implementací standardní knihovny C, která pracuje na libovolné architektuře s přidáním několika nízkoúrovňových rutin.
- GCC: Sada překladačů vytvořených v rámci projektu GNU. Původně se jednalo pouze o překladač programovacího jazyka C, později byly na stejném společném základě vytvořeny překladače jazyků C++, Fortran, Ada a dalších.
- GDB: Standardní nástroj na hledání chyb v GNU software. GDB podporuje mnoho programovacích jazyků, například C, C++ nebo Fortran.

2.5.3. Vývojové nástroj OpenOCD

OpenOCD[26] poskytuje ladění, interní programování a testování vestavěných zařízení. Tyto operace provádí pomocí JTAG (IEEE 1149.1) nebo SWD adaptéru. Jednou z jeho úloh je nahrávání binárních dat do zařízení (např. Flash paměť).

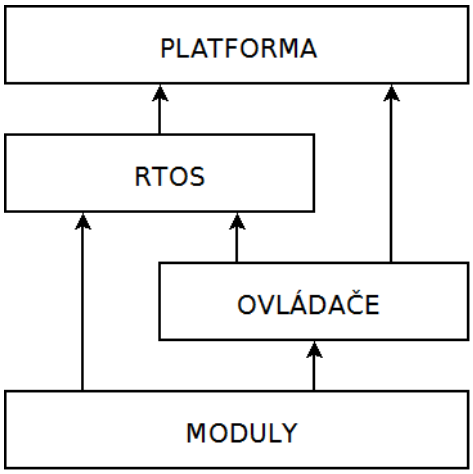
26 The Open On-Chip Debugger

3. Návrh a implementace

3.1. Kompilace a inicializace

3.1.1. Rozdělení systému

Rozdělení celého systému (firmwaru) na podsystémy (Obr. 15), nejlépe nezávislé, je velmi důležité. Usnadňuje to nejen orientaci ve zdrojových kódech, ale také vývoj a případné hledaní chyb. Pokud podsystémy běží nezávisle, je větší šance, že se případná chyba nepřenese i jinam a nezpůsobí pád celého systému. Tento způsob se dnes používá při mnoha jiných implementacích.



Obr. 15 Rozdělení systému

Základní funkce systému jsou rozdělené na moduly. Modul pracuje nad operačním systémem a nad ovladači (tzv. API). Modul nikdy nepřistupuje přímo k zařízení dané platformy, ale může přistupovat k ostatním modulům. Moduly si berou nejvíce procesorového času a všechny důležité funkce daného problému se provádí v nich.

Nad vším je postaven operační systém, v mém případě operační systém reálného času FreeRTOS. Ten zabezpečuje tvorbu a propojení všech vláken, které jsou součástí modulů. Poskytuje bezpečný plánovač a možnost přiřadit prioritu vláknům a tím i modulům řešící daný problém. FreeRTOS vlastní svůj port na danou platformu, proto nepoužívá ovladače.

Pro spojení s hardwarem dané platformy slouží ovladače. Některé jsou jednoduché, že zvládnou obsluhovat zařízení bez pomoci operačního systému a jiné tuto pomoc potřebují, například pro řízení přerušení. Při změně vývojové platformy se projeví největší změny v nich a samotné moduly zůstanou nezměněny.

Samostatnou a důležitou částí systému je vývoj webové služby. Tvoří část modulu webového serveru, ale její princip je běžet na vzdálené platformě. Daný klient si ze systému stáhne potřebná data a spustí je na své platformě, potom už díky propojení bude data jen aktualizovat.

Toto rozdělení se promítá i do rozdělení zdrojových kódů:

- *api* – obsahuje ovladače každé použité periferie
- *board* – obsahuje zdrojové kódy inicializace se základním nastavením
- *module* – obsahuje všechny použité moduly i včetně modulu pro jejich správu
- *rtos* – obsahuje zdrojové kódy FreeRTOS a portu pro danou platformu

V hlavní složce jsou jsou dále soubory *global.h* a *global.c* pro nastavení a funkce platné pro celý systém. Soubor *main.c* má hlavní funkci registrace modulu a zapnutí plánovače operačního systému.

Všechny soubory jsou dále děleny na hlavičky (*include*), zdrojové soubory (*source*), assembler (*assembler*) a připadne použité skripty (*script*).

3.1.2. Správa paměti

Paměť na platformě AT91SAM7X512 je velmi omezená. Skládá se z 128kB paměti pro data (RAM) a 512kB paměti pro program (ROM). K mikrokontroléru je připojená i další 512kB flash paměť, ale tato paměť nemůže být přímo adresovatelná a proto slouží jen jako dlouhodobé úložiště dat nebo nastavení. Případný další paměťový prostor může být zpřístupněný například pomocí USB připojení, to ovšem zatím nebylo implementováno.

Největším problém není nedostatek paměti pro program, ale nedostatek paměti pro data. Za celou dobu vývoje nebyla překročena hranice 256kB programové paměti. A i kdybych se v budoucnu dostával k hranici 512kB, tak kompilace pomocí Thumb instrukcí by mi přinesla další úsporu. Největším nebezpečím se tedy stala paměť RAM, která se při neopatrnosti může rychle přiblížit ke své hranici.

Pro kontrolu se při kompilaci vytváří soubor *CBS.map*, ve kterém lze zřetelně vidět spotřebu paměti jak pro program tak pro data. Obsah paměti pro program začíná na adrese 0x00100000 a obsah paměti pro data začíná na adrese 0x00200000.

Příklad řádku z *CBS.map*:

```
.bss            0x0020d2ac      0x3e94 ./rtos/source/heap.o
```

Zkratka .bss značí staticky alokované proměnné, 0x0020d2ac je adresa první alokace, 0x3e94 je délka alokované části a ./rtos/source/heap.o je soubor, který alokaci provádí. V tomto případě se asi jedná o alokování haldy pro systém o velikosti kolem 16kB.

Kontrola paměti ovšem nezaručí správné umístění všech dat na správné místo a také kompilátor potřebuje vědět kterou část RAM má použít, aby se data navzájem nemíchala nebo neztrácela. K správnému rozmístění slouží linkovací skript, který je uložen ve složce *board/script*. Pomocí něj se na sebe seskládají všechny druhy dat potřebné pro kompilaci (program, konstanty, globální proměnné, inicializované globální proměnné, zásobníky) nebo dokáže například vložit kousek programu i do RAM. Jeho hlavní úlohou je i kontrola zda zkompilovaný systém nepřesáhne hranici paměti.

Důležitou části je alokace paměti pro zásobníky, které slouží pro vytváření lokálních proměnných. Protože AT91SAM7X512 má sedm operačních režimů, tak pro každý z nich musí být stanoven vlastní zásobník. Tyto zásobníky jsou použity jen při inicializaci nebo například při přerušení, proto jejich velikost je malá nebo může být i nulová, pokud se daný režim nikdy nepoužije. Vlákna z FreeRTOS alokují svůj vlastní zásobník na haldě.

3.1.3. Vývoj a kompilace

Vývoj probíhá ve vývojovém prostředí Eclipse. V hlavní složce má nastavovací soubory pro nalinkování standardních knihoven. Eclipse byl použit i pro vývoj webové aplikace.

Kompilace se provádí pomocí kompilátoru v balíku YAGARTO. Protože celý kompilátor je odvozený od kompilátoru GCC, platí pro něj stejná nastavení (pár výjimek existuje). Během kompilace se vytváří spousta souboru, které slouží buď pro urychlení příští kompilace nebo jako výpisy pro vývojáře (soubory s příponou *.lst* je přepis do assembleru).

Celá kompilace je řízena pomoci programu Make a jeho nastavení se provádí v souboru *makefile* v hlavní složce. Soubor s nastavením je připraven pro přidávání nových zdrojových kódu bez jakýchkoli komplikací. Je zvolena vysoká míra optimalizace a při kompilaci se používá instrukční sada ARM. Výsledný binární soubor s názvem *CBS.bin* může být pomocí programu OpenOCD nahrán na programovou paměť zařízení (skripty pro nahráti jsou v hlavní složce).

3.1.4. Nastavení procesoru a postup při inicializaci

Jako mnoho jiných procesorů i AT91SAM7X512 spustí jako první instrukci tu na adrese 0x00000000. Adresy od 0x00000000 po 0x000fffff jsou v proměnné sekci a dají se přemapovat z ROM (0x00100000 na 0x00000000) nebo z RAM (0x00200000 na 0x00000000)). Jako výchozí je mapování nastaveno z ROM, proto jsou spuštěny instrukce nahraného systému (linkovací skript zajistil přesné umístění).

Většina zdrojových kódů je napsána v programovacím jazyce C, ale inicializace musí být napsaná v assembleru. Kdyby se použil jazyk C i pro inicializaci, není zaručeno že kompilátor nepoužije zásobník, který není v době spuštění procesoru definovaný. Inicializační soubory jsou ve složce *board*.

Postup inicializace:

1 Počáteční vektorová tabulka, skok je jen u prvního resetovacího vektoru 0 mířícího k pokračování inicializace, ostatní míří do nekonečné smyčky.

2 Nastaví dočasný zásobník a zavolá funkci *vBoardLowLevelInit*.

2.1 Vypne watchdog.

2.2 Nastaví časování EFC (typ AT91SAM7X512 má dva EFC).

2.3 Zapne hlavní oscilátor a nastaví hlavní takt na 48MHz.

2.4 Změna mapování na RAM (je mnohem rychlejší pro vektorovou tabulku). Během mapování jsou změněny některé vektory. Přerušení IRQ a FIQ míří do správce AIC a softwarové přerušení míří do FreeRTOS pro funkci plánovače.

3 Přemístí kód, který má být spouštěn v RAM, inicializuje nebo nuluje globální proměnné.

4 Projde všech sedm operačních módů procesoru a nastaví jim zásobník.

5 Nastaví operační mód na *Supervisor* (musí být, jinak FreeRTOS nepracuje) a zavolá funkci *main*

5.1 Vytvoří správce modulů.

5.2 Zaregistruje všechny zbylé moduly.

5.3 Spustí plánovač operačního systému.

6 Neočekává se návrat z *main*, nekonečná smyčka.

3.2. Ovladače

3.2.1. Flash kontrolér

EFC se stará o přístup k interní programové paměti flash. Slouží jak ke čtení instrukci pro procesor tak pro čtení dat. Zápis oproti čtení není zautomatizován a musí se provádět pomocí registrů. Zapisovat se dá vždy jen celá stránka (256 bytů) a složí k tomu zabudovaný zapisovací buffer. Do něj se mohou kopírovat jen slova (4 byty), takže buffer tvoří pole o 64 prvcích, které mají 4 byty.

Protože EFC používám jen pro ukládání nastavení, vybral jsem stránky na konci paměti, které jsou volné (začátek je obsazen programem).

Popis implementace:

Veřejná funkce sbEFCWrite

Jediná funkce v ovladači, která zapíše obsah zapisovacího bufferu do stránky poslané jako parametr. Kontroluje číslo stránky jestli existuje a také kontroluje jestli je paměť připravena pro zápis. Při zápisu do paměti flash se paměť předem nepromazává.

3.2.2. Ethernet kontrolér

EMAC se stal nejsložitějším ovladačem implementace. Jeho hlavní úlohou je komunikace s PHY čipem, který je připojený k mikrokontroléru. Čip PHY se dokáže automaticky přizpůsobit připojené síti, ale i přes to jsou potřeba některá základní nastavení (rychlost, duplex). Tyto nastavení se posílají přes managerské rozhraní. Čip obsahuje své vlastní registry, do kterých se zapisuje a čte. Jejich popis je v hlavičce *mii.h*. Na periferii EMAC je potřeba nastavit i MAC[27] adresu, pod kterou se

27 Media Access Control adresa, jedinečný identifikátor síťového zařízení

bude Ethernet rozhraní identifikovat.

Dále se definují takzvané deskriptory, do kterých se ukládají informace o přijatých nebo odeslaných rámcích Ethernetu. Jeden rámec se může skládat z více deskriptorů. Každý deskriptor má přidělen svůj buffer, ve kterém je uložena daná část rámce. Velikost bufferu pro příjem je dána na 128 bytů a pro odeslání jsem zvolil velikost 256 bytů. Deskriptory se postupně střídají podle toho jak jsou použity, například pokud byly použity pro jeden rámec deskriptory 1, 2 a 3 tak další rámec bude ve 4 a dalších deskriptorech. Pokud se deskriptory dostanou na konec svého počtu, pokračuje se od začátku. Nikdy se rámec nepřepíše přes druhy, protože se v deskriptoru nastavuje, jestli je připraven pro použití. Deskriptory nesou různá vedlejší data, například zda se mají přijímat broadcast rámce.

Protože přes EMAC se očekává velký datový tok, používá pro kopírování PDC. Jsou nastavena také dvě přerušení jak pro přijatý celý rámec a pro úspěšné odeslání celého rámce. Kromě synchronizace pomocí semaforu slouží tyto přerušení k zmiňovanému nastavení znovupoužití deskriptoru.

Popis implementace:

Neveřejná funkce vReadPHY

Funkce slouží pro čtení dat z managerského rozhraní. Definuje se adresa zařízení a čtený registr.

Neveřejná funkce vWritePHY

Funkce slouží pro zápis dat na managerské rozhraní. Definuje se adresa zařízení a na který registr se mají data zapsat.

Veřejná funkce sbEMACInit

Funkce slouží k inicializaci periferie EMAC. Po přivedení časování k periferii se k ní přidělí i dané piny. Důležitý je reset pomocí NRST pinu ke kterému je PHY čip připojen. Poté je zvolen MII[28] (existuje i RMII[29]) režim a zapne se časování zařízení MII. Nyní následuje nastavení deskriptorů a jejich bufferů. Je povolen příjem všech nebroadcastových rámců a do bufferu se nebude kopírovat FCS[30]. Po nastavení MAC adresy následuje nastavení PHY čipu přes managerské rozhraní. Na konec se vytvoří daná přerušení a semafor pro synchronizaci.

Veřejná funkce sbEMACRead

Funkce slouží ke čtení rámce z rozhraní, při čemž vrátí i jeho délku. První se zkontroluje jestli je zvolen správný deskriptor a poté je ze všech daných bufferu zkopírován rámec. Po zkopírování jsou dané deskriptory uvolněny pro další použití. Funkce nemá blokující charakter, takže pokud nenalezne žádný deskriptor, který nese rámec, vrátí chybu.

28 Media Independent Interface, standardní interface pro připojení Fast Ethernetu
29 Reduced Media Independent Interface, standardní interface pro připojení Ethernetu s redukovanými piny.
30 Frame Check Sequence, kontrolní byty na konci rámce.

Veřejná funkce sbEMACReadErase()

Funkce slouží k propadnutí rámce. Funkce je velmi podobná funkci *sbEMACRead*, ale neprovádí kopírování ale rovnou uvolní deskriptory pro další použití. Funkce se provádí, jestliže došla paměť a nemůže se rámec přijmout.

Veřejná funkce sbEMACWrite

Funkce slouží k zápisu rámce na rozhraní. První je zkontrolováno, jestli pro daný rámec dost volných deskriptorů. Pokud jsou volné, jsou deskriptory označeny jako použité a rámec je zkopírován do jejich bufferu.

Veřejná funkce vEMACWriteClear

Funkce je volána z přerušení. Uvolní všechny deskriptory použité pro poslední poslání rámce.

Veřejná funkce sbEMACInputWait

Blokující funkce, která slouží k zjištění zda přišel nový rámec.

3.2.3. Reset kontrolér

RSTC je správce resetu. Dokáže resetovat procesor, periferie, ale i zařízení připojené na NRST pin. U externího resetu se musí definovat čas, jak dlouho bude NRST pin udržovat resetující úroveň. Správce dokáže zapnout i uživatelský reset, kterým se dá resetnout procesor z vnějšího zdroje. To se často používá pro ladící programy.

Popis implementace:

Veřejná funkce vRSTCResetProcessor

Resetuje jádro mikrokontroléru (ukazatel instrukcí skočí na adresu 0x00000000).

Veřejná funkce vRSTCResetPeripheral

Resetuje všechny periferie mikrokontroléru.

Veřejná funkce vRSTCResetExternal

Funkce slouží k resetu zařízení připojených na NRST pin. Je definovaná délka resetu a funkce cyklicky čeká dokud není reset dokončen.

Veřejná funkce vRSTCResetUser

Funkce slouží k zapnutí nebo vypnutí uživatelského resetu.

3.2.4. Čítač reálného času

RTT slouží pro počítání času od určitého období. Jeho přesnost je nastavena na 0.0625s, což znamená, že první nejnižší 4 bity 32-bitového čítače slouží jako část jedné sekundy a zbylé bity pro počet sekund. Nastavení děličky se automaticky přizpůsobuje podle aktuální rychlosti procesoru. I když tato technika nastavení není nutná, protože procesor jede na krystalu (velice přesný). Pokud by ale jel pouze na RC oscilátoru, jeho přesnost by se měnila teplotou a napětím. Předpokládaná rychlost hlavního oscilátoru je vynásobena šestnácti a poté vydělena aktuální rychlostí procesoru načtenou z registru. Tím se získá hodnota pro děličku, když se požaduje 1s. Pokud se požaduje menší čas, musí se vydělit daný vzorec zpřesňující hodnotou (pro 0.5s je 2, pro 0.25s je 4 a pro 0.0625 je 16).

Popis implementace:

Veřejná funkce vRTTInit

Funkce slouží k zapnutí (resetu) čítače a je nastaven na danou přesnost 0.0625s.

Veřejná funkce vRTTRead

Funkce slouží k přečtení aktuálního času z čítače a vložení do 64-bitové proměnné. Sekundy jsou uloženy do vyšších 32 bitů a část sekundy je v nižších 32 bitů.

3.2.5. Sériové periferní rozhraní

SPI umožňuje sériovou komunikaci s připojenými zařízeními, v tomto případě spojení s flash pamětí AT45DB041D. Flash paměť má dostatečnou rychlostní rezervu (dokáže vyšší rychlost než procesor), proto je dělička SPI nastavena na minimální hodnotu (takt jádra je 48MHz, dělička 2). Pro přeposílání dat z a na rozhraní je použito PDC. Synchronizace s přerušení je provedena pomocí semaforu.

Popis implementace:

Veřejná funkce sbSPIInit

Funkce slouží k inicializaci rozhraní. Po přivedení časování k periferii se k ní přidělí i dané piny. Zapne se master mód a nastaví se daná dělička. Vybere se řídící pin, ke kterému je flash paměť připojena. Následuje vytvoření semaforu a přerušení.

Veřejná funkce sbSPIWrite

Funkce slouží k nastavení zapisovacích bufferu pro PDC. Po jejich nastavení jsou data odeslána.

Veřejná funkce sbSPIRead

Funkce slouží k nastavení čtecích bufferu pro PDC. Také zapne přerušení pro zjištění zda jsou

už buffery zaplněny.

Veřejná funkce sbSPIWaitRead

Funkce slouží k blokování dokud nejsou přijata všechna očekávaná data.

3.2.6. Synchronní / asynchronní sériové rozhraní

Ovladač USART spravuje dvě sériové rozhraní, které jsou navzájem nezávislé. Každé z těchto rozhraní může být nastaveno na dva režimy: RS-232 režim pro spojení s GSM modemem a RS-485 režim po pro spojení se Síťovou ochranou. Oba tyto režimy jsou přímo podporované periferii USART. Výběr režimu pro dané rozhraní se volí při inicializaci, ale pokud byl režim pro jedno rozhraní zvolen, druhé rozhraní musí mít jiný režim než první rozhraní (například obě rozhraní nemůžou sloužit v režimu RS-232).

RS-232 slouží k posílání příkazu a dat do a z modemu GSM. Při tomto přeposílání se využívá hardwarový handshake, který striktně přikazuje také použití PDC. Problémem u použití PDC je, že se musí vědět, jaká velikost dat bude přijímána. Tento nedostatek je vyřešen časovou kontrolou mezer mezi přijímanými bity. Tato kontrola nastaví čítač při každém přijatém bitu a pokud čítač vyprší naskočí přerušení. Tímto je nadefinovaná minimální vzdálenost pro rozpoznání konce dat.

RS-485 slouží k posílaní příkazů a dat do a z NSU modulů. Na této sběrnici se provozuje protokol Modbus, proto nechybí spočítání CRC i jeho kontrola u přijatých dat. Ovladač umí přeposílat data na více modulů dle dané adresy (každá adresa má svůj vlastní synchronizační semafor), ale žádný systémový modul toto nevyžívá a ani to nebylo otestované. RS-485 nevyužívá PDC jako režim RS-232, data jsou pomocí front a přerušení posílána po jednom bytu na rozhraní. Tento způsob je pomalejší, ale pro malé množství dat je dostačující.

Popis implementace:

Neveřejná funkce rGetBaudrate

Funkce slouží k přepočtu požadované rychlosti. Výpočet se přizpůsobuje nadefinované rychlosti procesoru a správně zaokrouhluje hodnoty pro větší přesnost.

Neveřejná funkce vCRC16

Funkce slouží pro výpočet kontrolního součtu CRC s řídícím polynomem stupně 16.

Veřejná funkce sbUSARTInit

Funkce slouží k inicializaci jednoho vybraného rozhraní. Kromě nastavení samotného rozhraní (režim, rychlost, parita) se vytvářejí potřebné synchronizační semafory nebo fronty. Nakonec je nadefinovaná i minimální časová hodnota pro zjištění konce dat.

Veřejná funkce sbUSARTRS485Socket

Funkce slouží pro zpřístupnění adresy předané jako parametr. Vytvoří se fronta pro příjem na dané adrese. Tato funkce slouží pouze pro mód RS-485.

Veřejná funkce sbUSARTRS485Write

Funkce slouží k zápisu dat na rozhraní RS-485. Automaticky se použije standard Modbus. Před daty je vložena daná adresa NSU zařízení a za daty je vložen kontrolní výpočet CRC. Funkce je chráněna mutexem.

Veřejná funkce sbUSARTRS485Read

Funkce slouží ke čtení dat z rozhraní RS-485. Také se automaticky použije standard Modbus. Má blokující charakter a čeká určitý čas na příchozí data. Pokud data nedojdou, je vrácena chyba. Příchozí kontrolní výpočet CRC je porovnán se spočítaným. I tato funkce je chráněna mutexem.

Veřejná funkce sbUSARTRS232Transmit

Funkce slouží ke komunikaci s rozhraním RS-232. Složí jak k posílání tak k příjmu dat. Pro komunikaci je použito PDC a pro rozeznání konce příchozí zprávy je použit semafor, který je sesynchronizovaný s přerušením.

3.3. Moduly

3.3.1. Správce modulů

Jeden z nejdůležitějších modulů, který se stará o ostatní moduly. Jeho hlavní úlohou je jejich inicializace a případné zrušení (zatím se nepoužívá). Také se stará o udržování jejich aktuálního stavu:

- NONE – modul neexistuje
- STOP – modul právě stojí, tento stav je ihned po registraci modulu nebo po smazání
- INIT – modul se právě inicializuje
- DELETING – modul se právě zastavuje a maže
- RUN – modul běží
- SUSPEND – modul je pozastaven (není implementováno)
- ERROR – modul má kritickou chybu

Případný druh kritické chyby je také zaznamenán:

- NONE – kritická chyba neexistuje
- INIT – kritická chyba v inicializaci ovladače
- MEM – kritická chyba v alokování paměti

Protože moduly mají být nezávislé, předávají správci modulu svá různá varování. Tyto varování poté mohou ostatní moduly využít pro svou činnost. Seznam varujících zpráv:

- NONE – varování neexistuje
- CONNECTION – varování s připojením (např. rozpojené rozhraní)
- TIME – varování při špatné synchronizaci reálného času
- INTERRUPTION – varování s rušením při přenosu (např. špatné CRC)
- START – varování při nabíhání mikrokontroléru (nenajely všechny požadované moduly)

Pro rozeznání každého modulu dostane každý z nich modulové identifikační číslo (MID). Tímto číslem se mohou moduly dotazovat na své stavy nebo je i nastavovat. Všechny dotazovací funkce jsou chráněny mutexem.

Popis implementace:

Vlákno vModuleThread

Vlákno slouží pouze k zavolání funkce *ubModuleStartAll* a poté se samo uspí. Toto vlákno jako jediné běží v plánovači operačního systému po jeho spuštění.

Neveřejná funkce ubModuleStartMID

Funkce slouží k projití inicializačního procesu daného modulu. Během inicializace je modul ve stavu INIT a po jeho úspěšném spuštění se stav změní na RUN. Funkce je chráněná mutexem.

Neveřejná funkce ubModuleStopMID

Funkce slouží k projití mazacího procesu daného modulu. Během mazání je modul ve stavu DELETING a po jeho úspěšném ukončení se stav změní na STOP. Funkce je chráněná mutexem.

Veřejná funkce ubModuleInit

Funkce slouží k inicializaci správce modulů. Je spuštěn už během inicializace mikrokontroléru před spuštěním plánovače operačního systému. Správce se zde také zaregistruje jako modul s MID 0. Je zde vytvořeno vlákno *vModuleThread*, které se spustí hned po rozjetí plánovače.

Veřejná funkce ubModuleRegistration

Funkce slouží k registraci nového modulu. Registrace se provádí před spuštěním plánovače operačního systému. Předává se inicializační funkce, která se bude volat a funkce pro zastavení. Funkce pro zastavení je dobrovolná, modul může být nesmazatelný za běhu. Je možnost zvolit jestli se modul má spustit ihned po najetí plánovače nebo bude spuštěn později manuálně. Modul má i svůj název, podle kterého ho lze vyhledat ve správci.

Veřejná funkce ubModuleStart

Funkce slouží k rozjetí modulu dle jeho jména. Funkce je chráněná mutexem.

Veřejná funkce ubModuleStartAll

Funkce slouží k rozjetí všech modulů, které mají povolen start ihned po najetí plánovače. Funkce je chráněná mutexem.

Veřejná funkce ubModuleStop

Funkce slouží k zastavení modulu dle jeho jména. Funkce je chráněná mutexem.

Ostatní veřejné funkce

Ostatní funkce slouží k získaní jména z MID nebo MID z jména. Také slouží ke čtení a nastavení varování a chyb modulů. Stavy modulů lze jen číst, jejich nastavení probíhá automaticky. Všechny funkce jsou chráněny mutexem.

3.3.2. Správce paměti

Tento modul se stará o přístup k paměti, která buď nemá přímo adresovatelný přístup (externí flash paměť) nebo její zápis je podmíněn sekvencí příkazů (interní flash paměť). Modul je ovšem stále ve vývojové fázi a stále mění svou podobu během testování a nalézání chyb. Zatím modul umí spravovat konfiguraci systému, která se ukládá na interní flash paměť. Přístup k těmto datům je pomoci ukazatele, který míří na začátek stránky z nastavovacími údaji. Během inicializace modulu je zkontrolováno jestli nebyla celá flash paměť smazána (všechny buňky budou obsahovat hodnotu 0xff). Pokud je smazána, načtou se výchozí hodnoty do zapisovacího bufferu a stránka se uloží, k čemuž slouží ovladač EFC.

Popis implementace:

Veřejná funkce vStorageInit

Funkce slouží k nastavení výchozích hodnot jestli je to potřeba.

Veřejná funkce sbStorageConfWrite

Funkce slouží k uložení dat ze zapisovacího bufferu do dané stránky. Jako parametr se předává reálná adresa stránky.

Veřejná funkce sbStorageConfSetDefault

Funkce slouží k zapsaní právě uložených hodnot na dané stránce do zapisovacího bufferu té samé stránky.

3.3.3. Vrstva TCP/IP

Jádro TCP/IP modulu je tvořeno zdrojovými kódy knihovny lwIP. Knihovna řeší vše od TCP spojení až po použití DHCP klienta. V konfigurační hlavičce *lwipopts.h* lze změnit nastavení všech částí knihovny. Velký důraz je kladen hlavně na konfiguraci paměti, protože knihovna může mít obrovské nároky jak na datovou tak i na programovou paměť. Knihovna má v jádru své vlastní vlákno, které řeší všechny důležité operace.

Soubor *lwip.c*

Po zavolání funkce *vlwIPInit* správcem modulů je inicializována knihovna lwIP. Poté je přidán nový interface, při čemž se zvolí zda se má spustit DHCP klient nebo se nastaví statické adresy (IP, maska, brána, DNS). Po inicializaci je interface zapnut a data mohou být přijímána. Pro změnu adres za běhu slouží funkce *sblwIPResetSetting,* která načte nové data z flash paměti.

Soubor *sys_arch.c*

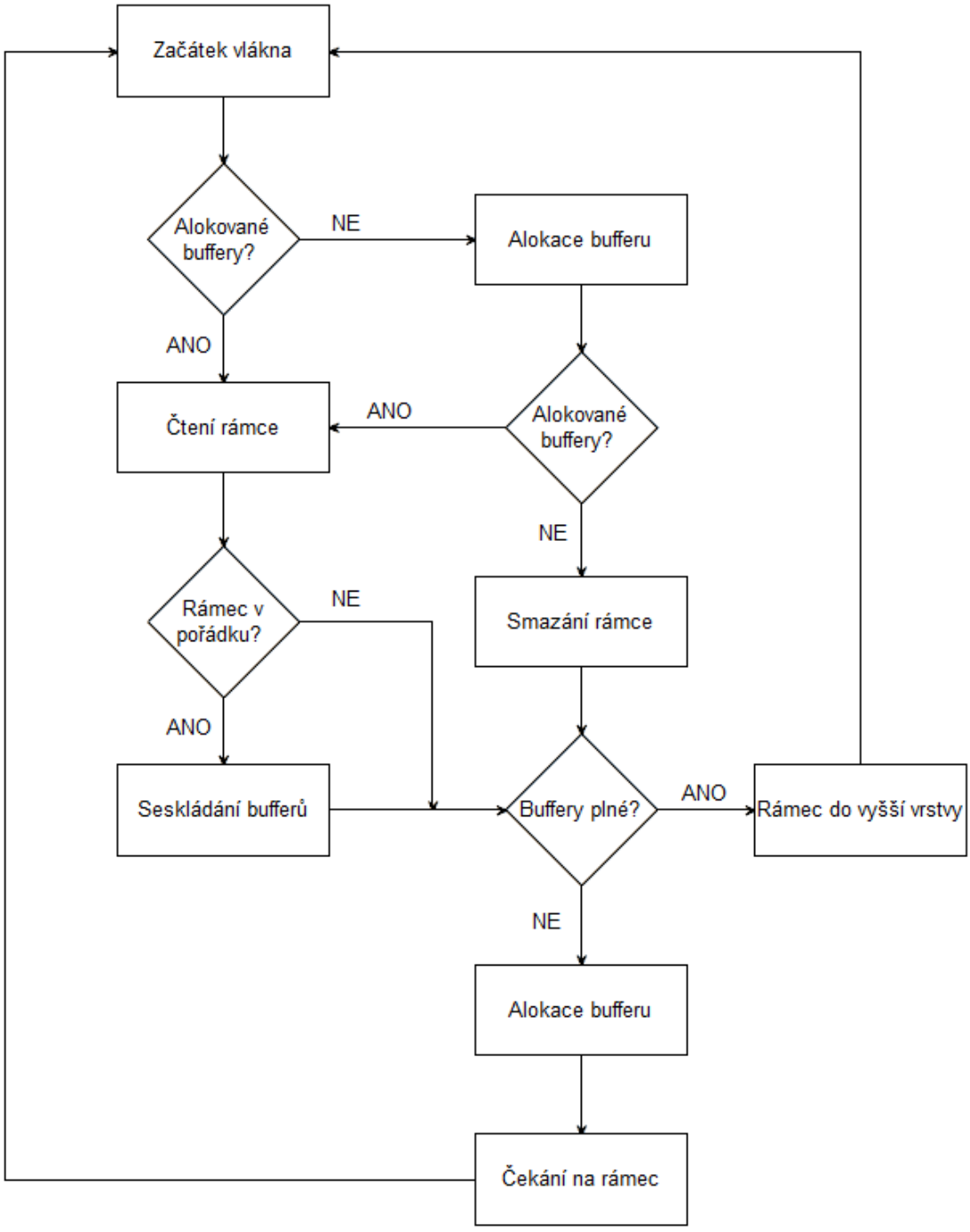
Další části je port knihovny. Knihovna dokáže pracovat s vlákny a používat synchronizační nástroje a právě v souboru *sys_arch.cjsou* všechny potřebné funkce.

Soubor *ethernetif.c*

V souboru se nachází spojení mezi knihovnou lwIP a ovladačem EMAC. Soubor se dá rozdělit na tři částí:

1. První část se věnuje příjmu rámců a jako jediná část je tvořena samostatným vláknem. Zjednodušený algoritmus je zobrazen na následujícím diagramu (Obr. 16). Základem jsou předvytvořené buffery, kterých je tolik aby se do nich vlezl jeden rámec. Jejich velikost byla po pozorování nastavena na 256 bytů (nesmí se plýtvat pamětí a také nesmí být zbytečně velký počet bufferu). Pokaždé jak přijde rámec, zaplní jeden až maximum bufferů. Ty jsou poté propojeny a poslány do vyšších vrstev, kde ho převezme hlavní vlákno knihovny. Před dalším čtením se pokusí znovu alokovat chybějící buffery, pokud se to nepovede další příchozí rámec je smazán.

   Tato část je tvořena funkcemi *ethernetif_input*, *ethernetif_input_alloc* a *low_level_input*.



Obr. 16 Vlákno pro příjem rámců Ethernetu

2. Část zabývající se odesíláním rámců. Hlavní vlákno knihovny pošle několik bufferu, které budou rámec obsahovat. Tyto buffery jsou prověřeny a jejich obsah je poslán do ovladače EMAC.

   Tato část je tvořena funkcí *low_level_output*.

3. Část, která se stará o inicializaci rozhraní i ovladače EMAC. Tato část je volána knihovnou. Vytváří vlákno pro první část a alokuje buffery. Také přiřazuje MAC adresu ovladači EMAC a pro knihovnu (použití v ARP[31]).

   Poslední část tvořena funkcemi *ethernetif_init*, *ethernetif_input_alloc* a *low_level_init*.

3.3.4. Webový server

Hlavní funkci firmwaru má být provozování HTTP serveru. Protože existuje mnoho druhu platforem, které se k systému budou přihlašovat, bylo nutné striktně dodržet standard HTTP 1.1 alespoň v jeho minimální podobě. Důležitou podmínkou při implementaci bylo jednoduché a přehledné přidávání dalších zdrojů do webového serveru (jak statických stránek tak i generovaných dat).

Server používá knihovnu lwIP pro vytvoření TCP spojení mezi klienty a systémem. Pro připojení ke klientům se používá API z třídy netconn, které nejsou složité a zvládají synchronizaci vláken. Pro přístup k příchozím bufferům se používá API z třídy netbuf. Nevýhodou lwIP knihovny je možné rozdělení příchozích dat do několika bufferu, proto jsou všechny data nakopírovány do HTTP bufferu. Tento buffer se později použije i k vytváření nestatických dat pro odeslání.

Pokaždé když je přijatá nějaká žádost (GET, POST, HEAD) je vytvořena struktura, do které se budou zapisovat všechny informace v průběhu řešení žádosti. Pokud nastane nějaká známá chyba odpoví server danou zprávou klientovi. Server zvládá také privilegovaný přístup ke zdrojům.

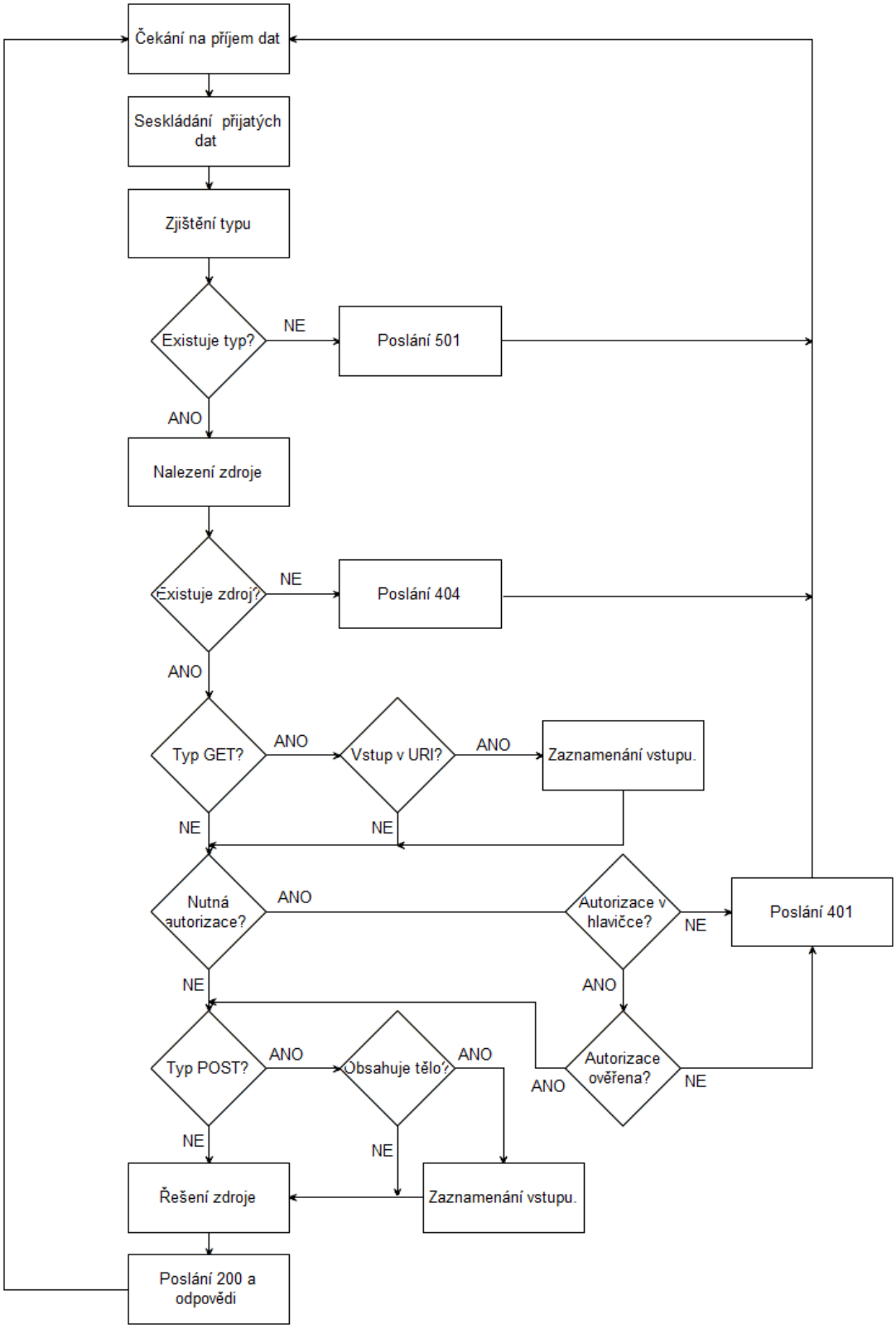
Server je rozdělen na pět částí, každá z ní řeší jiný problém:

1. Hlavní část tvořena souborem *http.c*. Zde kromě inicializace, ve kterém se spustí hlavní vlákno, je řešeno spojení s TCP vrstvou. Algoritmus hlavního vlákna je na následujícím diagramu (Obr. 17).

   Příchozí data jsou z lwIP bufferů nakopírována do HTTP bufferu. Vzhledem k možnosti poslání jedné HTTP žádosti ve více segmentech, čeká HTTP určitou dobu, zda jsou všechna data přijata.

   Poté započne řešení žádosti. První je zjištěn typ (GET, POST, HEAD), pokud typ není implementován, je vrácena chyba klientovi. Další části je vyhledání zdroje a to se provádí ve druhé části HTTP modulu. Pokud není zdroj nalezen je zase vrácena chyba klientovi. Pokud je žádost typu GET, následuje ověření zda v URI nejsou data (pro POST se provádí později). Pokud ano, jsou nalinkované do struktury pro pozdější vyřešení.

31 Address Resolution Protocol, používá se k získání Ethernetové MAC adresy v počítačových sítí



Obr. 17 Vlákno HTTP modulu

Na řadě je procházení hlavičky a kontrola autentizace. Pokud se ve struktuře objeví požadavek na autentizaci, pokusí se jí najít mezi hlavičkami. Pokud není nalezena, pošle se chyba klientovi. Pokud autentizace nalezena je, pošle se autentizace do části tři. Pokud autentizace prošla touto části s chybou je opět poslána zpráva klientovi. V hlavičce se také zjišťuje délka těla žádosti, pokud nějaké tělo existuje. Pokud tělo existuje, je nalinkované do struktury (pouze pro POST).

Následuje už jen zavolání řešení z částí dvě a poslat z něj klientovi odpověď.

2. Druhá část je tvořena seznamem zdrojů a jsou tu i jejich řešící funkce. Pro vyhledání zdroje slouží funkce *sbHTTPResourceSearchData*. Funkce se pokusí o nalezení zdroje se shodným jménem. Pokud je zdroj nalezen, jsou do struktury uloženy všechny potřebné informace pro další zpracování (řešící funkce, potřebná autorizace).

   Pokud hlavní vlákno serveru dojde až k řešení zdroje, je zavolána jedna z funkcí z této části. Protože statické data se řeší všechny stejně, mají stejnou řešící funkci *sbHTTPResourceStatic*, která vybere zvolená data z části tři. Pro řešení dynamických dat (forma JSON) jsou zde ostatní řešící funkce.

   Tato část se řeší v souboru *http-resource.c*.

3. Třetí část se řeší v souboru *http-auth.c a* obsahuje pouze kontrolu uživatelského jména a hesla (skrývají se za hashovacím algoritmem).

4. V souboru *http-data.c* jsou pouze uložena statická data pro odpovědí na žádosti.

5. Poslední část řeší tvorbu odpovědí a to jejich první řádek nebo těla chybových zpráv. Je zde seznam všech používaných odpovědí, jejich číslo i text. Pro tvorbu odpovědi jsou zde i funkce. Tato část se řeší v *http-status.c*.

Jak už jsem psal výše, přidání dalšího zdroje je jednoduché. Stačí vytvořit zdroj a řešící funkci (nebo použít stávající) v souboru *http-resource.c* a pokud je to nutné, vytvořit statická data v souboru *http-data.c*.

3.3.5. Klient SNTP

Modul SNTP řeší získání odpovědi od vzdáleného NTP serveru. Pro spojení využívá UDP vrstvu z lwIP knihovny. Používá z ní i funkci pro překlad jmen, která získá IP adresu z jména NTP serveru. Kromě inicializace, která připraví UDP spojení, modul obsahuje funkci *sbSNTPGetTime*. Tato funkce vrací čas v 64-bitové hodnotě získaný od NTP serveru.

Popis implementace:

Veřejná funkce sbSNTPGetTime

Funkce se připojí k NTP serveru dané adresy a poté mu pošle danou strukturu dat. Před posláním je do struktury vložen aktuální čas (byty reprezentující čas musí být otočeny). Po přijetí je

zkontrolováno jestli odpověď opravdu poslal dotazovaný NTP server. Pokud tomu tak je, je ze struktury zkopírován nový čas (zase se musí otočit byty).

3.3.6. Modul reálného času

Tento modul spolupracuje s modulem SNTP, ale také potřebuje k práci ovladač RTT. Modul se stará o reálný čas v systému a snaží se ho udržet synchronizovaný. Aktuální čas je uložen na dvou místech. První je 64-bitová hodnota určující čas poslední synchronizace. Tato hodnota je definovaná jako čas od roku 1900. Druhou hodnotu udržuje ovladač RTT ve svém čítači. Tato hodnota je definovaná jako čas od poslední synchronizace. Po sečtení obou 64-bitových čísel vznikna aktuální reálný čas.

Popis implementace:

Vlákno vRTimeThread

Vlákno řešící aktualizaci reálného času. První část vlákna spočítá časovou odchylku dle vzorce z kapitoly 2.3.1, při čemž mu poslouží modul SNTP k získání aktuálního času ze serveru NTP. Pokud vypočítaná odchylka přesáhne 60 sekund nebo nebyl ještě načten žádný čas, dojde k velké chybě v synchronizaci. Při této chybě je ovladač RTT restartován a hodnota poslední synchronizace násilně nastavena na hodnotu získanou z NTP serveru.

Druhá část vlákna slouží k pomalé synchronizaci času, pokud je odchylka menší jak 60 sekund. To se provádí pomocí urychlování nebo zpomalování čítače v ovladači RTT (míra byla určena pozorováním). Po restartu ovladače RTT s aktualizovaným čítačem se uloží i nová hodnota poslední synchronizace.

Veřejná funkce vRTimeInit

Funkce slouží k inicializaci modulu. Volá se tu i inicializace SNTP, protože není spuštěn správcem modulů.

Veřejná funkce sbRTimeGetTime

Funkce slouží pro získání aktuálního reálného času. Vrácený čas je zde také posunut na období od roku 1970. Funkce je chráněná mutexem.

Veřejná funkce sbRTimeGetStringTime

Funkce slouží k získání aktuálního času ve tvaru řetězce. Splňuje tvar, který se používá v hlavičkách zpráv ze standardu HTTP 1.1.

3.3.7. Modul GSM

Tento modul používá ovladač USART pro přístup k rozhraní RS-232. Jeho hlavní úkolem je

udržování spojení se zařízením GSM a možnost posílat přes něj SMS zorávy do GSM sítě. Spojení je udržováno pomocí opakujících se dotazů. Pokud z těchto dotazů nedostane odpověď, oznámí to správci modulů.

Celý algoritmus běží ve vlákně vytvořeném ve funkci *vGSMInit*. V ní je i inicializace rozhraní RS-232 na daném USARTu. Diagram vlákna je na následujícím obrázku (Obr. 17). Použité dotazy jsou v kapitole 2.2.1. Po každém z těchto dotazů je očekávaná odpověď ve tvaru poslaného příkazu plus znaky OK<CR><LF>.

Pro poslání SMS slouží funkce *sbGSMSendSMS*, která je chráněna mutexem. Jako parametr funkce se předává tělo zprávy a číslo, na které se má zpráva poslat. Z těchto parametrů vytvoří dotaz na GSM zařízení a pomocí bufferu a fronty předá požadavek do vlákna GSM. Ten, pokud má spojení se zařízením, zprávu odešle.

3.3.8. Modul síťové ochrany

Jednoduchý modul pro příjem dat od zařízení Síťová ochrana. Pro přeposílání dat přes RS-485 využívá ovladač USART. Podobně jako předešlé moduly obsahuje i tento funkci pro inicializaci, ve které se spustí ovladač USART a zpřístupní se Modbus adresa daného zařízení. Pro uložení dat slouží dva buffery, z jednoho se může číst aktuální data, na druhý se mezitím zapisují příchozí odpovědi. Modul obsahuje dvě řešící vlákna:

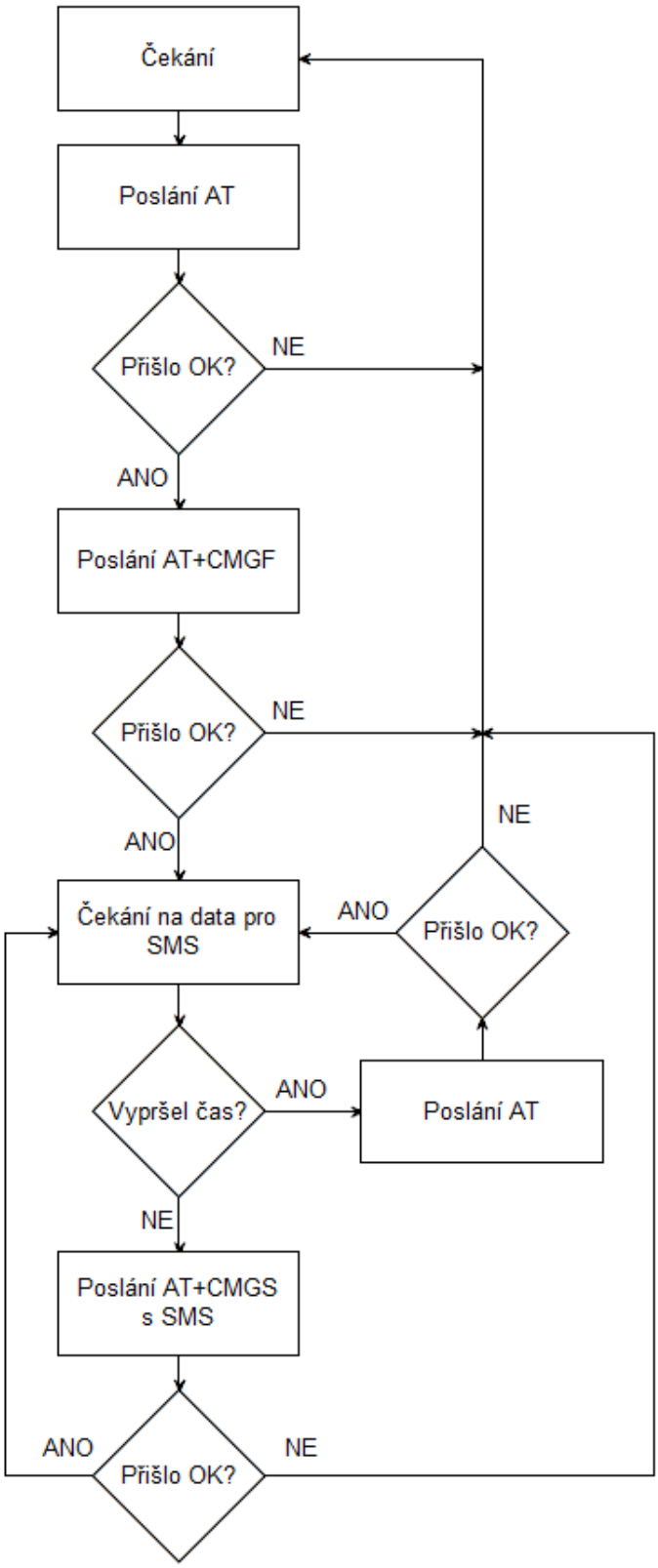
1. První vlákno *vNSUThreadTx* pouze cyklicky přeposílá požadavek o data na zařízení.
2. Druhé vlákno *vNSUThreadRx* čeká na odpověď a ukládá postupně data do bufferu pro zápis. Po přečtení všech dat je buffer vyměněn s bufferem pro čtení aktuálních dat, příští zápis se provede do něj. Pokud nepřichází odpověď v určitém intervalu, oznámí správci modulů že zařízení není připojeno.

3.4. Webové rozhraní

3.4.1. Grafické rozvržení

Webová aplikace je vizuálně viditelné prostředí pro daného klienta. Každý klient má jiné požadavky, a proto stránky zde uvedené slouží jen jako šablona.

Pro tvorbu této stránky byl použit jak základní značkovací jazyk HTML tak i jazyk pro kaskádové styly CSS. Tyto dva jazyky tvořili základní vzhledovou kostru. Příklad jednoduché kostry je na obrázcích (Obr. 18, Obr. 19). Tato kostra má tři základní sekce: Přehled, Nastavení a O systému. Tyto základní sekce se dělí na podsekce, například u sekce Přehled: Události, Stavy, Síťová ochrana, Grafy.

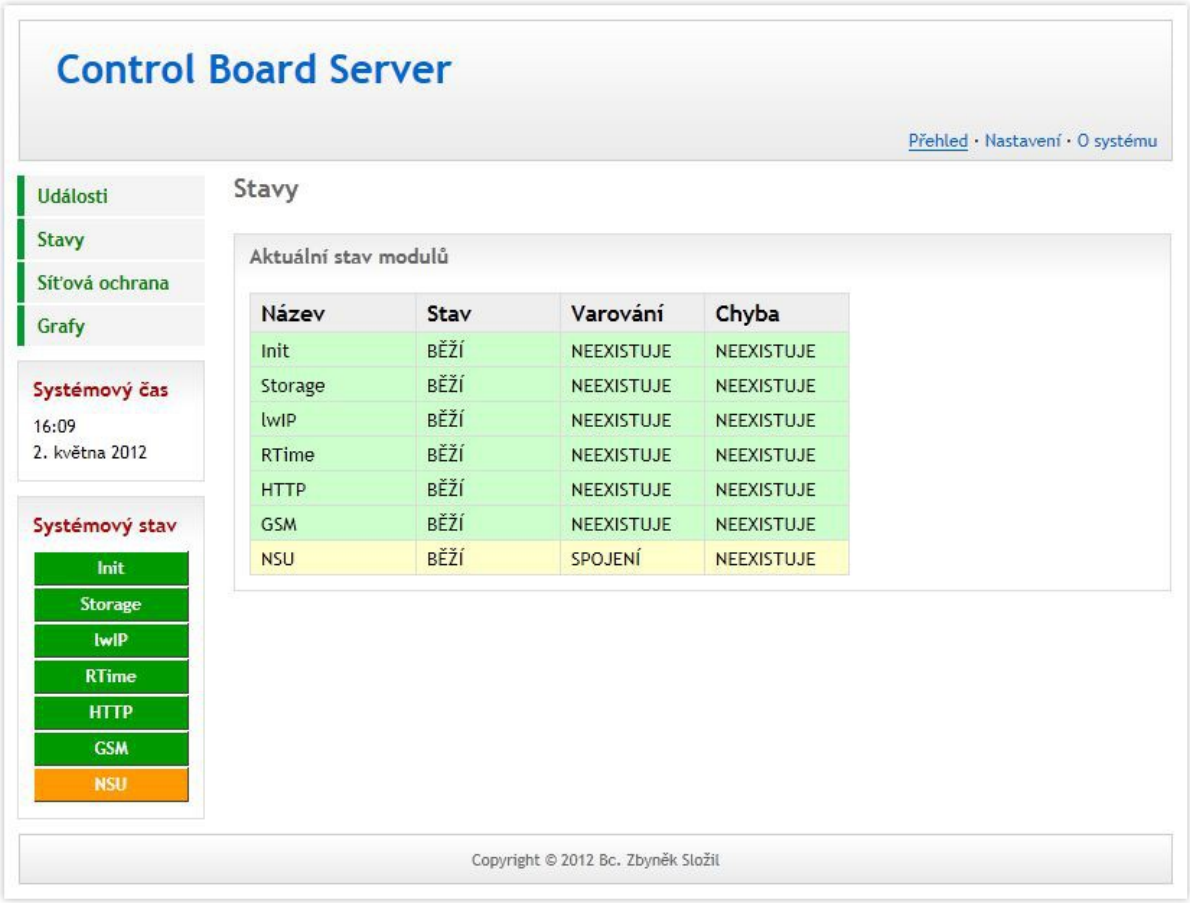


Obr. 17. Vlákno GSM modulu

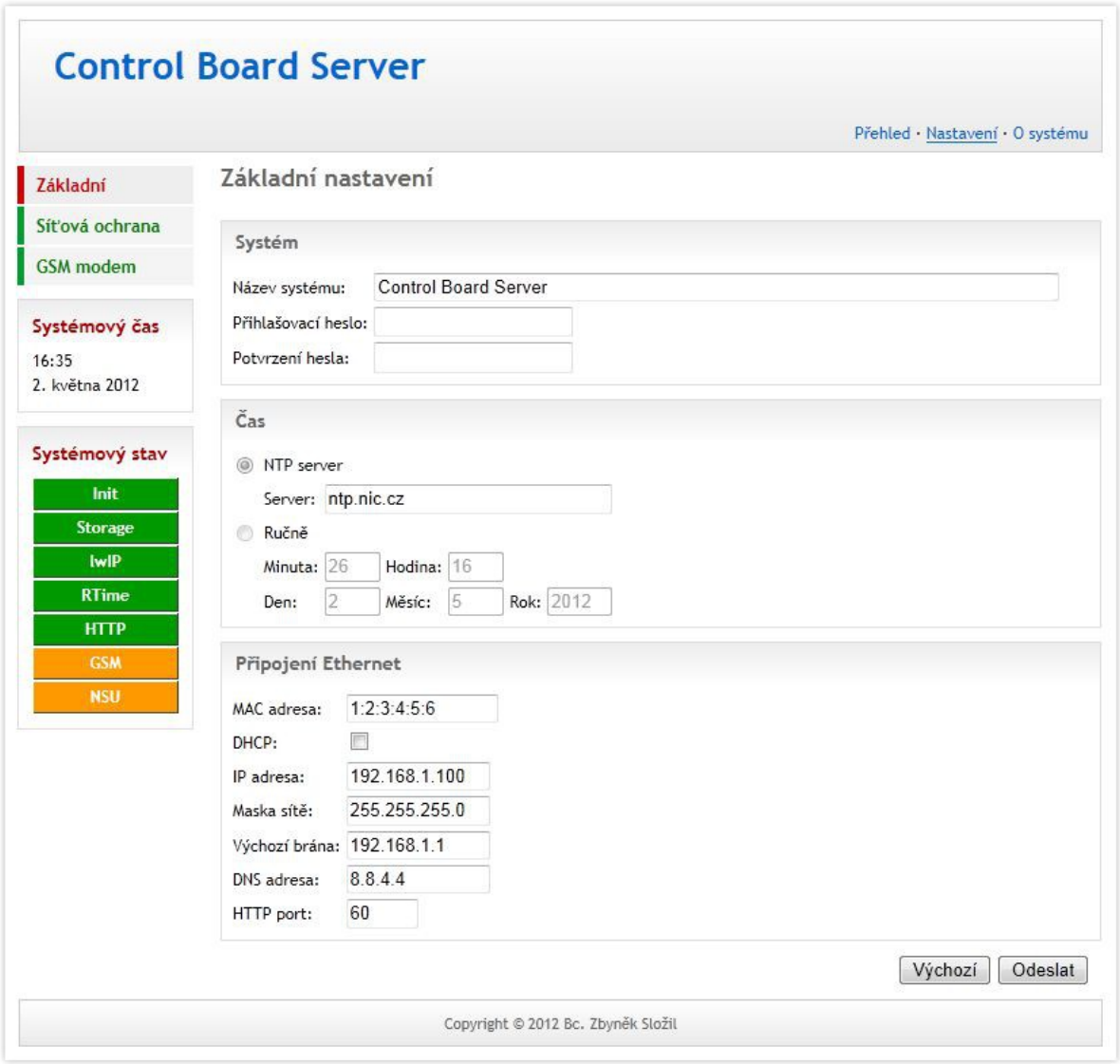
3.4.2. Dynamickým obsah

Dynamický obsah je ze systému posílán ve formátu JSON a pomocí programovacího jazyka JavaScript zpracován a vložen do grafické kostry. Tyto skripty jsou pouštěny buď cyklicky, aby byl obsah udržován aktuální (čas, stavy), nebo jsou puštěny jednou, například pro získání aktuálního nastavení systému (Obr. 19).

Při řešení nastavení a posílání dat do systému nebyl použit základní formulář HTML, ale pouze funkce JavaScriptu, která vstupní data zkontrolovala regulárními výrazem (označí se špatně vyplněné kolonky). Poté jsou data poslána do systému a pokud došlo k úspěšnému uložení, jsou ty samé data vrácena zpět.



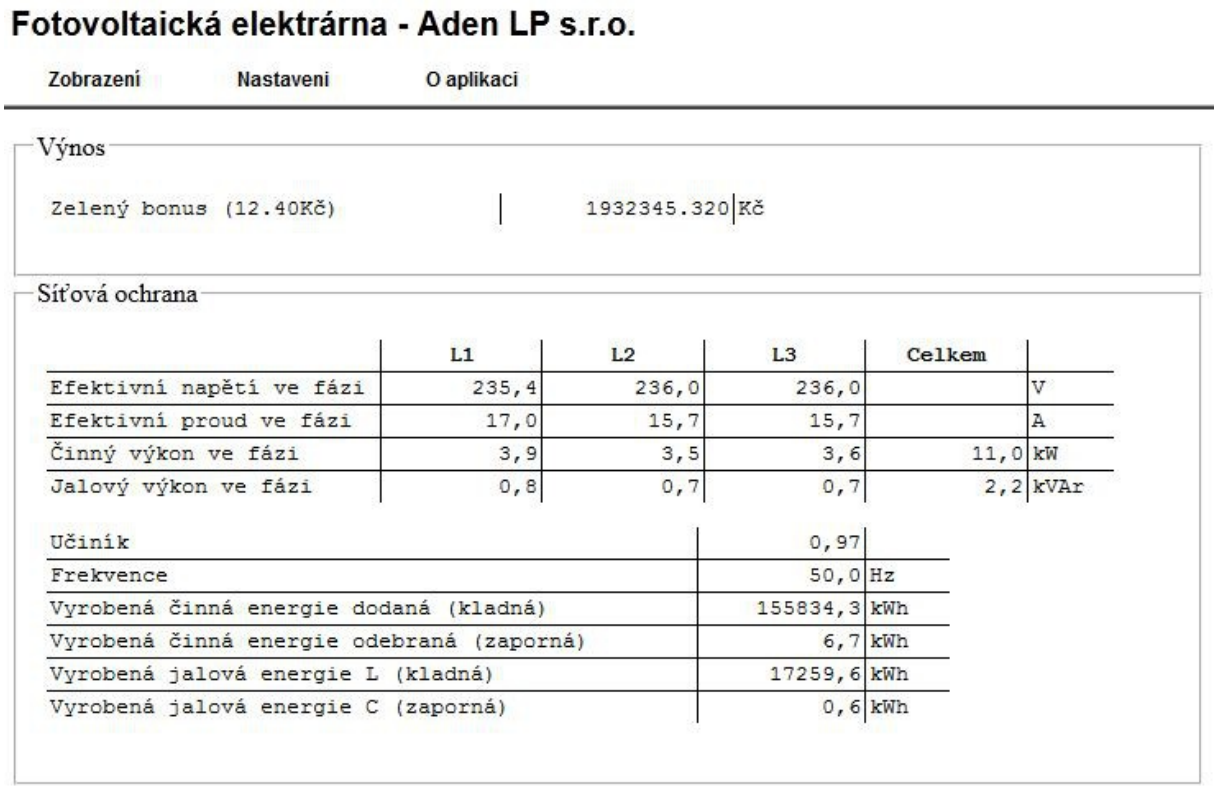
Obr. 18 Webové rozhraní - Stavy



Obr. 19 Webové rozhraní - Nastavení

4. Testování a známé chyby

Testování systému bylo prováděno postupně a každý modul prošel jinou úrovní těchto testů. Testovala se hlavně funkčnost a spotřeba paměti. Některé moduly nebo jejich části se dostaly i do reálného provozu (modul NSI, Obr. 20) a jiné moduly byly předělány nebo úplně vyměněny. Testování nového HTTP modulu (na Obr. 20 je použitá starší verze) bylo prováděno hlavně v domácím simulovaném prostředí.



Fotovoltaická elektrárna - Aden LP s.r.o.

Zobrazení Nastaveni O aplikaci

Výnos

Zelený bonus (12.40Kč)	1932345.320	Kč

Síťová ochrana

	L1	L2	L3	Celkem	
Efektivní napětí ve fázi	235,4	236,0	236,0		V
Efektivní proud ve fázi	17,0	15,7	15,7		A
Činný výkon ve fázi	3,9	3,5	3,6	11,0	kW
Jalový výkon ve fázi	0,8	0,7	0,7	2,2	kVAr

Učiník	0,97	
Frekvence	50,0	Hz
Vyrobená činná energie dodaná (kladná)	155834,3	kWh
Vyrobená činná energie odebraná (zaporná)	6,7	kWh
Vyrobená jalová energie L (kladná)	17259,6	kWh
Vyrobená jalová energie C (zaporná)	0,6	kWh

Obr 20. Reálný provoz starší testovací verze

V projektu existuje mnoho mě známých chyb, které se musí opravit.. Mnohé z těchto chyb pochází i z použitých knihoven a je třeba konzultace s danými vývojáři. Například chyba, při které knihovna lwIP z neznámého důvodu ukončuje TCP spojení ještě před přijetím HTTP žádosti (proto se nenačtou všechny obrázky v prohlížeči). Chyba byla sic nalezena a způsob opravy jsem zatím nezjistil.

Také nastavení pomocí webového rozhraní nepracuje správně. Někdy je nutné restartovat mikrokontrolér aby se změny nastavení projevili.

5. Závěr

Protože projekt byl můj, rád bych na něm pracoval i nadále. Celý jeho vývoj trval necelé dva roky, kdy jsem se učil pracovat s danou architekturou a také hledal, zkoušel užitečné knihovny. Tyto knihovny se staly největším úskalím, protože mnohé z nich jsou zastaralé nebo nemají dostatečně popsané zdrojové kódy (luštění tisíce řádku cizího kódu je zdlouhavé). Ale i přes to informace a zkušenosti, které jsem za dobu tvorby tohoto projektu získal, jsou pro mě velmi cenné.

Nynější implementace také nemá zpracované všechny části, které jsem očekával. Hlavně bych rád dokončil úschovu naměřených hodnot a jejich zobrazení na webovém rozhraní. Jejich reprezentaci v grafech by také mohlo být přínosem.

Projekt je dostatečně univerzální a může být upraven na kontrolu mnoha jiných systémů. Pokud se opraví všechny kritické chyby a vytvoří se další potřebné moduly může spolehlivě sloužit i několik let. Stále ale zůstává hlavní použití ve fotovoltaických elektrárnách nebo kontrole různých elektrických generátorů.

6. Literatura

[1] *Co se děje v počitači* [serial online]. Česká republika: root.cz [cit. 2012-05-04].

[2] ATMEL. *AT91SAM ARM-based Flash MCU* [online]. 2011 [cit. 2012-05-04].

[3] *JSON*. [online]. [cit. 2012-05-04]. Dostupné z: http://www.json.org/

[4] *The FreeRTOS Project* [online]. [cit. 2012-05-04]. Dostupné z: http://www.freertos.org/

[5] *Představujeme FreeRTOS*. [online]. [cit. 2012-05-04].
Dostupné z: http://www.mcuhobby.cz/2010/01/predstavujeme-freertos/

A. CD/DVD

Obsah CD/DVD:

- Text diplomové práce
- Zdrojové kódy k programu